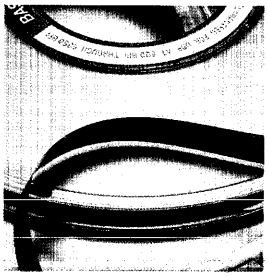
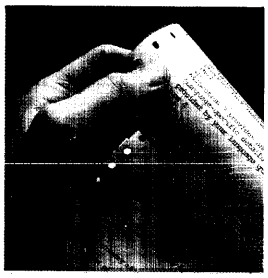
Prime Computer, Inc.

DOC10001-1PA

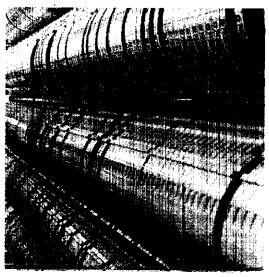
Software Release Document

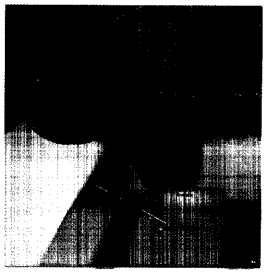
Rev. 19.4

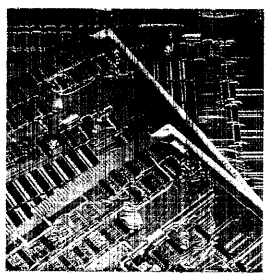













Software Release Document

Revision 19.4

by

Jacki Forbes

and

George W. Gove

This guide documents the software operation of the Prime Computer and its supporting systems and utilities as implemented at Master Disk Revision Level 19.4 (Rev. 19.4).

Prime Computer, Inc.
Prime Park
Natick, Massachusetts 01760

COPYRIGHT INFORMATION

PRINTING HISTORY — Software Release Document

Edition	Date	Number	Software Release
First Edition	April 1985	DOC10001-1PA	Rev 19.4

CUSTOMER SUPPORT CENTER

Prime provides the following toll-free numbers for customers in the United States needing service:

1-800-322-2838 (within Massachusetts)
1-800-343-2320 (within other states)
1-800-541-8888 (within Alaska)
1-800-651-1313 (within Hawaii)

HOW TO ORDER TECHNICAL DOCUMENTS

Obtain an order form, a catalog, and a price list from one of the following:

Inside U.S.

Software Distribution
Prime Computer, Inc.
74 New York Ave.
Framingham, MA 01701
(617) 879-2960 X2053

Outside U.S.

Contact your local Prime subsidiary or distributor.

CONTENTS

5 DATA MANAGEMENT SYSTEMS

ABOUT THIS BOOK

This book summarizes the new features and changes in Prime's user software at Rev. 19.4 of PRIMOS. One chapter is devoted to each of the following:

- Executable Program Formats (EPFs) (Chapter 2)
- PRIMOS and Utilities (Chapter 3)
- Languages (Chapter 4)
- Data Management Systems (Chapter 5)
- INFORMATION (Chapter 6)
- FORMS and FED (Chapter 7)
- Communications (Chapter 8)

Within each chapter, the information on each product (e.g. COBOL, MIDAS, RJE) begins on a new right-hand page. Pages can thus be extracted from this book and placed in other manuals as necessary.

Note

This book is designed to supplement other manuals. Its pages are not replacement pages, and its pagination does not correspond to the pagination of other books.

For each individual product, this book provides the following information (where applicable):

- New features and changes in the software
- Documentation corrections and additions
- Software problems fixed
- Software problems outstanding

In all cases, this information refers to Rev. 19.4. If you want information on earlier versions of Rev. 19, see:

- The Rev. 19.3 Software Release Document (MRU4304-011)
- The Rev. 19.2 Software Release Document (MRU4304-010)
- The Rev. 19.1 Software Release Document (MRU4304-009)
- The Rev. 19.0 Software Release Document (MRU4304-008)

Note

This book contains corrections and additions to other Prime software manuals. It is assumed that you have access to our most recent documentation. Lists of new books and updates can be found in the following places.

- The section New Book Titles, in Chapter 1, for titles new at Rev. 19.4.
- The Rev. 19.3 Software Release Document for titles new at Rev. 19.3.
- An online file, accessible by typing HELP DOCUMENTS, for a cumulative list of manuals, updates, etc.

CHAPTER 1

INTRODUCTION

IN THIS CHAPTER

This chapter provides:

- An overview of Rev. 19.4
- Information on installation of Rev. 19.4
- A list of new book titles from Prime's Technical Publications Department

OVERVIEW OF REV. 19.4

Rev. 19.4 contains many new features of interest to Prime users, operators, and Administrators. The following lists only the highlights, arranged according to the intended audience.

New Features for the User

The following will interest the Prime user:

- Executable Program Formats (EPFs)
- BIND, Prime's new linker

- Search rules
- An enhanced Command Environment, and several new commands to utilize and control the environment
- Changes to the Spooler Subsystem
- Auto Speed Detect

EPFs: EPFs (Executable Program Formats) are dynamic runfiles (programs) that are assigned by PRIMOS at runtime to any segments. EPFs can be suspended and re-invoked without loss of data by running the EPFs in different "command levels" of PRIMOS. The number of command levels, number of program invocations per command level, and number of private static and private dynamic segments you can have is limited by the System Administrator via EDIT_PROFILE. EPFs are created with BIND, Prime's new linker. For more information on EPFs see Chapter 2. EPFs are more fully documented in the Programmer's Guide to BIND and EPFs and in the Advanced Programmer's Guide.

BIND: Prime's new linker, BIND, creates dynamic runfiles (or EPFs). BIND's subcommands can be entered on the command line or BIND can be run interactively. For more information on BIND, see Chapter 2. BIND is fully documented in the Programmer's Guide to BIND and EPFs.

Search Rules: At Rev. 19.4 there is a new file, ENTRY$.SR, in the directory SYSTEM. This required file contains names of system libraries, one per line. The order in which these libraries are listed is the order in which the libraries will be searched to find a match to the subroutine entrypoints. You can define your own library search list in a file in your own directory and request that PRIMOS follow the system list with your own list.

New Commands: Several new commands have been added at Rev. 19.4 to support EPFs and library search lists. They are:

INITIALIZE_COMMAND_ENVIRONMENT
: Restore your command environment to the state it is in when you first log in.

LIST_EPF
: List information about any or all of the EPFs you are currently using.

LIST_MINI_COMMANDS
: List those commands you can use if you reach Mini Command Level.

LIST_LIBRARY_ENTRIES
: List the entrypoints in library EPFs.

LIST_LIMITS
: Display your command environment limits.

LIST_SEARCH_RULES
Display the contents of your library search list.

LIST_SEGMENT List segments you are using.

REMOVE_EPF Remove an EPF from your address space.

SET_SEARCH_RULES
Set your own search rules.

These commands are described in Chapter 2.

Enhanced Command Environment: At Rev. 19.4 you are no longer limited to a maximum of 127 file units per user. Because the maximum number of file units per system has been increased from 3247 to 32,762, there is no longer a need to be concerned with an individual's number of reserved file units.

Spooler Subsystem Enhancements: The Spooler can now handle lines up to 400 characters in length. However, T$LMPC, the parallel printer driver, still truncates lines to 140 characters.

Several new options have been added to the SPOOL command to allow you to modify your own print requests and to list files spooled to remote spool queues. The options are:

-DISK Allows you to specify the partition name of the SPOOLQ you wish to use.

-LIST There are three new options available:

* list all spool queues on all partitions that have been added to your system.

AT list only the SPOOLQ specified by AT.

OWN list only your own entries in the queue.

-MODIFY Modify your own print request.

-NOTIFY Tells the Spooler phantom to send a message to your terminal when your print request has printed.

-TRUNCATE Print only the first n characters in each line.

These options are described in Chapter 3.

Auto Speed Detect (ASD): If the line to which you are connecting is ASD enabled, and if your terminal generates either "mark" or "odd" parity, PRIMOS can now determine the following terminal line speeds automatically: 110, 300, 1200. PRIMOS can also determine the

following terminal line speeds if your System Administrator has configured your system to do so: 2400, 4800, and 9600.

New Features for the Operator

The following will interest the Prime operator:

- Disk Controller enhancement
- Changes to the Spooler Subsystem
- Changes to the CHAP command

Disk Controllers: Controllers at addresses '22 and '23 can now contain paging partitions. These controllers are accessible under PRIMOS II, however, you cannot use MAKE for these controllers under PRIMOS II.

Changes to the Spooler Subsystem: There are three new options to the SPOOL command that are of interest to the operator. They are:

-MODIFY Allows you to change the parameters of a print request.

-RUSH Allows you to give a file high priority so that it will be printed before other files in that SPOOLQ.

-NORUSH Cancels the priority given previously with the -RUSH option.

Changes to the PROP Command: A new option has been added to the PROP command.

-COMPRESS Renumbers the entries in a designated spool queue and removes gaps between entries in the queue.

Changes to the CHAP Command: IDLE and SUSPEND are two special-purpose new priority levels. A process CHAPped to IDLE will be serviced only when no other process requires service from the CPU. SUSPEND blocks the suspended process from access to the CPU until it is explicitly CHAPped otherwise.

Changes for the System Administrator

Changes to Configuration Directives: There are four new configuration directives:

LOGBAD	Message will be printed at the supervisor terminal if an unsuccessful login attempt is made.
MEMHLT	Under certain circumstances will log out a process incurring an uncorrectable memory parity error instead of halting the machine.
NVMFS	Defines the system-wide number of Virtual Memory File Access segments for EPFs.
SYNC	Can be used as a synonym for SMLC.

The FILUNT directive has changed because the number of segments per user has changed to a maximum of 32,762 ('77772). It is no longer necessary to reserve file units on a per user basis.

The NUSEG directive has been replaced by EDIT_PROFILE subcommands at Rev. 19.4.

MONITOR_NET: The MONITOR_NET command replaces the MONITOR_RING command. MONITOR_NET displays error and traffic information about RINGNET, synchronous lines, and virtual circuits for your system. You can select any one of three monitors (RING, SYNCHRONOUS LINE, or VIRTUAL CIRCUIT) from the Main Menu. Each of these monitors has an overview screen and one or more detailed screens.

Changes to Synchronous Support: At Rev. 19.4, dial-up half duplex support for Prime-to-Prime links is now available over MDLC.

256 Lines Supported: At Rev. 19.4, up to 256 lines can be connected to a system. The maximum line number is '377 (255 decimal). These lines may exist on AMLC, ICS1, or ICS2 controllers. No more than 128 lines may be on AMLC controllers. For ICS2 controllers, the async line adapter cards may be distributed over up to four ICS2s, even if the total number of ICS2 lines is less than 256. Lines '376 and '377 can only be used as assignable lines if the formula user number = line number + 2 is maintained.

EDIT_PROFILE: Four attributes have been added to EDIT_PROFILE at Rev. 19.4 to define a user's (or a project's) command environment.

These attributes allow specification of the following characteristics.

- Number of command levels
- Number of live program invocations per command level
- Number of private static segments
- Number of private dynamic segments

Changes to the Spooler Subsystem: Revision 19.4 supports a number of new Spool subsystem features. It now fully supports Prime's Office Automation System. Multiple SPOOLQ directories are now allowed on one system. You can now choose between two algorithms for how a Spooler phantom on a networked system searches SPOOLQs for print requests. You can define those users who should be able to modify the prop environment in a new file, L.USER. These features are described in Chapter 3.

INSTALLING REV. 19.4

The Rev. 19.4 Master Disk contains all Master Disk software, including the softwear that has not changed since Rev. 19.0. Rev. 19.4 can therefore be installed without prior installation of other Rev. 19 versions.

If you are installing from tape, you should delete the following files before restoring the tapes.

CMDNC0>BOOT_ATTACH.CPL	HELP*>FTOP.HELP	LIB>SPLLIB.BIN
CMDNC0>BOOT_IMPCODE.CPL	HELP*>FTGEN.HELP	LIB>PFTNLB.BIN
CMDNC0>BOOT_TREE.CPL	HELP*>FTS.HELP	LIB>NPFTNLB.BIN
CMDNC0>BOOT_CREATE.CPL	HELP*>FTR.HELP	LIB>NSPLLIB.BIN
CMDNC0>BOOT_SAVE.CPL		

Because the new files are of a different file type, MAGRST will not restore them over another file of the same name.

The PRIMOS.COMI file and Shared Segments: Check your PRIMOS.COMI (or C_PRMO) file against the file PRIMOS.COMI.TEMPLATE in the PRIRUN directory of the Master Disk Revision 19.4 software. FTNLIB, COBOL, and SPLLIB are now EPFs and are no longer shared using the SHARE command.

NEW BOOK TITLES

The following technical publications are new at Rev. 19.4.

PRIMOS and Utilities

The System Operator's Guide, Volume II has been broken up into the following series of books, designed for ease of use.

DOC9298-1LA,1PA Operator's System Overview
DOC9299-1LA,1PA Operator's Guide to System Monitoring
DOC9300-1LA,1PA Operator's Guide to File System Maintenance
DOC9301-1LA,1PA Operator's Guide to System Backups
DOC9302-1LA,1PA Operator's Guide to the Batch Subsystem
DOC9303-1LA,1PA Operator's Guide to the Spooler Subsystem
DOC9304-1LA,1PA Operator's Guide to System Commands

For operators of Prime computers. These books assume some knowledge of how to start up and use a Prime computer.

This series of books was designed and written to help operators and System Administrators of Prime computer systems do their jobs. The series covers the techniques and commands necessary for the smooth functioning and operation of Prime computers to facilitate the tasks of the systems' users.

DOC8691-1LA,1PA Programmer's Guide to BIND and EPFs

For all users of Prime computers who write programs in compiled, high-level languages (such as COBOL and FORTRAN) or in assembler (PMA), all of which require the use of a linker. This book assumes that the reader knows how to write and compile a program in a Prime-supplied language and has read the Prime User's Guide.

This book introduces the user to BIND, a linker new at Rev. 19.4, and Executable Program Formats (EPFs), the type of runfile generated by BIND. The differences between BIND and its predecessors, SEG and LOAD, are examined, as are the differences between EPFs (which are dynamic runfiles) and static runfiles generated by SEG and LOAD.

The following series of books has been written for the Advanced programmer.

DOC9229-1LA,1PA Advanced Programmer's Guide, Volume 0
DOC10055-1LA,1PA Advanced Programmer's Guide, Vol. I: BIND and EPFs
DOC10056-1LA,1PA Advanced Programmer's Guide, Vol. II: File System
DOC10057-1LA,1PA Advanced Programmer's Guide, Vol. III: Command Environment

The Advanced Programmer's Guide is a series of books that deals with system-level programming concepts in-depth. Most of the information

pertains specifically to the 50 Series, although some topics apply to operating systems or computing systems in general.

For Rev. 19.4, the series consists of four volumes. Volume 0 describes the notation used in other volumes in the series and contains appendixes on new features for Rev. 19.4 that are particularly important for a systems-level programmer and on PRIMOS error codes and their meanings. The other three volumes examine specific topics in detail: BIND and EPFs, the File System, and the Command Environment.

This series is for users of Prime computers who write programs in compiled, high-level languages (such as PL/I Subset G and FORTRAN) or in assembler (PMA), all of which require the use of a linker. This series assumes that the reader knows how to write, compile, link, and execute a program written in a Prime-supplied language as an EPF using BIND, and that the reader has read the Prime User's Guide, Programmer's Guide to BIND and EPFs, and the Subroutines Reference Guide. Portions of this series also assume that the reader understands the Prime 50 Series architecture, particularly the procedure call, virtual memory, shared memory, memory segmentation, and dynamic linking mechanisms.

The following updates to existing documents are available:

PRIMOS and Utilities

DOC4130-4LA,4PA Prime User's Guide
FDR3108-5LA,5PA PRIMOS Commands Reference Guide
DOC5037-2LA,2PA The System Administrator's Guide

Languages

UPD3621-31A Subroutines Reference Guide
UPD5040-11A RPG II Programmer's Guide
UPD5039-11A COBOL 74 Reference Guide
UPD3058-13A BASIC/VM Reference Guide
UPD4031-13A PL/I subset G Reference Guide
UPD4303-21A Pascal Reference Guide
UPD4029-31A FORTRAN 77 Reference Guide
UPD7534-11A C Reference Guide
UPD4033-21A Source Level Debugger User's Guide

Data Management

UPD6292-11A DBMS Administrator's Guide
UPD7345-11A Data Management File Administrator's Guide
UPD5308-11A DBMS DML Reference Guide
UPD5717-11A DBMS DDL Reference Guide
DOC7999-2LA,2PA PRISAM User's Guide, Rev. 19.4

Communications

UPD3710-31A PRIMENET Guide
DOC7532-2LA,2PA Network Planning and Administration Guide
UPD4035-21A Distributed Processing Terminal Executive Guide

CHAPTER 2

EXECUTABLE PROGRAM FORMATS

NEW FEATURES AND CHANGES

EXECUTABLE PROGRAM FORMATS (EPFs)

The introduction of EPFs and the enhanced command environment makes it easier for programmers to build and maintain software. The enhanced command environment provides programmers with new tools with which to build small, medium, and large programs. These tools are tailored towards increasing the capabilities of programs and may also increase the programming speed of developers.

This chapter presents software developers and System Administrators with an overview of the EPF product. This chapter also discusses differences between programming in the old environment (only static mode programs) and programming in the new environment (EPFs and an enhanced command environment).

This chapter discusses the various aspects of the EPF product. The order of the sections is designed to take the programmer through the various steps needed to understand EPFs and then to program with EPFs. The final section relates information that a System Administrator will need when installing the new system. The sections are:

- A brief overview of EPFs and the enhanced command environment
- A discussion of building and running EPFs
- A discussion of the enhanced command environment

- A discussion of programs calling programs
- A discussion of EPF libraries
- An explanation of the new commands that allow full utilization of the enhanced command environment
- A discussion of new user attributes
- A discussion of programming restrictions
- Information for the System Administrator
- A discussion and examples of converting existing static mode programs to EPFs.

For the beginning programmer, the sections on building and running an EPF are all that is needed. For the programmer who is concerned with putting together software packages, the sections on EPFs, Enhanced Command Environment, New User Attributes, and Error Detection, Handling, and Recovery are appropriate. For the programmer who will be converting static mode programs to EPFs, Programming Restrictions and Converting Old Static Mode Programs To EPFs will be useful. System Administrators should read the section discussing System Administrator Information.

A Brief Overview Of EPFs

There are two basic kinds of EPFs, the program EPF and the library EPF. A program EPF is an executable object that is invoked with RESUME. A library EPF is an executable object that contains entrypoints that are dynamically linked at run time. Library EPFs cannot be invoked with RESUME.

EPFs are not statically bound to segments via LOAD or SEG as are static mode programs. Prime's new linker, BIND, produces EPFs, and is much friendlier than either SEG or LOAD. (An entire BIND session can be performed from the command line.) BIND is described in the section EPFs - The New Program Format.

As BIND is easier to use than both SEG or LOAD, so EPFs are easier to use than static mode programs. Since EPFs take care of all addressing, the programmer need not remember where a program, or a structure within a program, is placed. Because the operating system takes care of loading EPFs into memory, it is now possible to have many different programs in memory at the same time. The operating system makes sure that these programs do not overwrite each other.

Command writing is easy with EPFs. Because EPFs run on the same stack as the command processor, it is easy for the command processor to execute EPFs. Only a simple procedure call instruction is needed. This means that EPFs appear to the operating system to be no more than another subroutine. This, in turn, makes it possible to pass arguments to and from a command.

A Brief Overview Of The Enhanced Command Environment

The enhanced command environment brings three new major features to PRIMOS:

- The ability to create personalized libraries of entrypoints. This aids the developer in both the development and maintenance phases of a project. It also makes it possible to put together software packages in a manner which was, up until this time, impossible.

- The ability to create commands which can directly interface with the PRIMOS command environment. These new commands can be directly passed from the command line entered by a user and can return their status directly to the command environment. In addition, command functions can now be written by the programmer.

- The ability to have programs call other programs. It allows programmers to write a transaction processor which runs user chosen transactions as separate programs which can be debugged as separate pieces.

As part of the enhanced command environment, special user attributes were developed. These attributes help a System Administrator to manage a system's resources. For further information, consult the section on New User Attributes.

Since EPFs run in a set of segments called dynamic segments (which is part of a user's DTAR2 segment), more segments have been added to DTAR2. A user can now be assigned up to 512 DTAR2 segments.

EPFs - The New Program Format

When Prime introduced processors containing full virtual addressing capabilities (paging and segmentation), the need to generate and use the V-mode code which supports this architecture was fulfilled by the language translators and SEG. The translators generate object code that can be loaded by SEG. This loaded image is known as a run file. It is this file which can be invoked by RESUME. In many cases, SEG needs special instructions about where and how to organize this file. With the advent of EPFs, the need for these instructions vanishes. The object code generated by the translators is no longer loaded together into a run file. Instead, BIND puts the object code into a form which is loaded by the operating system at run time.

In the past, Prime had two program formats, the R-Mode image and the SEG image. An R-Mode image is created by either LOAD or by SEG. (For SEG to create an R-Mode image, special SEG commands are needed.) The SEG image may only be created by SEG. The R-Mode image may be invoked with RESUME. The SEG image may only be run from SEG. Both of these program formats are static mode formats. All of their addressing

requirements, with the exception of dynamic linking, are fulfilled before they are executed. EPFs bring a new program format. This program format is built using BIND. Its image is known as a RUNfile. This file may be invoked with RESUME. The RUN file is a dynamic mode format. Many of its addressing requirements are not fulfilled until it is executed.

This chapter contains all the information that directly relates to the construction and types of EPFs, using, building, and executing EPFs, and programming with EPFs.

What Is An EPF?

The executable program format (EPF) implements a new program object representation for V- and I-mode programs. Unlike both SEG files and R-Mode SAVE files, which represent a memory image of both procedure and data, an EPF is a file that contains both an image of the procedure and a description of the data (linkage) for a given program. This description can be thought of as a set of templates that the operating system uses when it RESUMEs the EPF to build the linkage and resolve the addressing requirements. Since it is doing address resolution at execution time, the operating system places the EPF in a segment of the user's address space that it finds free. This capability allows the operating system to keep many EPFs in memory at the same time; EPFs will not write over each other.

DAM files were chosen to contain EPFs because their structure (indixes which point to the data records of the file) has two important features. The first is the speed with which the data records of the file can be found. One only has to go through an index to find any data record. The second has to do with how the operating system reads the data from an EPF file.

EPFs are special with regard to the manner in which the operating system reads their data. The operating system takes the information in the DAM index and stores it in the paging system data bases. Since paging data bases relate to where on disk the virtual memory for a page may be found, remembering the DAM index means that a mapping is made between the EPF file and the virtual memory it will occupy. More simply put, EPFs are directly paged in from their file system partition. No special copy is made on the paging partition as is done for a static mode file. This paging from the file system is known as Virtual Memory File Access or VMFA. One restriction is placed on this new form of paging: the address space associated with the EPF must be read only.

Since EPFs may contain read-only data only, they must not contain any code or data areas which need to be modified. In other words, EPFs must be pure code. Pure code is reentrant and EPFs may contain only pure code. The system takes advantage of this fact by mapping the same file to different users' address spaces whenever different users execute the same EPF; that is, EPFs are shared.

Since an EPF contains only pure code and templates which describe the linkage associated with the EPF, the operating system must dynamically allocate space in which it can build the actual linkage for the EPF. It does this at initialization time. All linkage for an EPF is initialized when it is first placed into memory by the operating system. Since the system keeps EPFs mapped into memory even after the EPF completes, the next time the EPF is to be executed, only the linkage, which needs to be set up for every run, will be initialized. No action will be taken for the rest of the linkage. This gives an advantage to EPFs that are executed many times, such as the LD command: they run faster. On the other hand, you may lose some initialization which SEG did for you; make sure you follow the language rules closely.

When an EPF is built, no instructions about where the stack for the EPF is to be placed are made. This is left up to the operating system. The system uses the same stack for the EPF that it is currently running on. Specifically, the system uses its command processor stack for EPFs. An EPF will look just like any other subroutine to the system. This is handy to know when debugging an EPF.

EPF Types

There are two types of EPFs:

- The program EPF is executed via a main entrypoint.
- The library EPF contains entrypoints to which dynamic links may be made.

Program EPFs

You may invoke Program EPFs with RESUME. Program EPFs have one main entrypoint to which flow of control is passed. These are the typical program images which you build.

You use a Program EPF whenever you would have used a SEG program. The enhanced command environment allows one program to "call" another program: a running program may RESUME another run image. If the programs calling each other are EPFs, a series of sequential calls can be made. Many EPFs may reside in memory at the same time without overwriting each other. However, only one static mode program may appear in this sequence of calls.

Although EPFs are shared between users, program EPFs may not be treated like some shared static mode programs such as the current version of EMACS. This is because programs like EMACS contain two types of linkage, per-user and per-system. The per-system linkage is shared and the per-user linkage is not. Program EPFs support only per-user linkage.

EPF programs may also be executed as commands or command functions. A command interface for EPFs has been defined allowing EPFs to pass information back to the command environment.

Simple EPF Calling Sequence: You may be building EPFs to run your own programs that do not need any information at initial execute time. These programs should use the following calling sequence:

```
dcl your_epf entry();
call your_epf;
```

Many programs that have already been written would use this calling sequence. An example is a TIC-TAC-TOE program. This program has no need of any user supplied information before it starts. It simply puts up the playing board on the screen and then asks the player for a move.

EPF Command Calling Sequence: You may be writing programs that need information from the command line. That is, whenever anyone invokes these programs, they will enter added information on the command line. The EPF product will automatically pass this command line information to the invoked program. We call these programs commands. Additionally, the EPF product also allows the command to pass its status back to the operating system whether or not it has completed successfully. The calling sequence for commands follows:

```
dcl your_epf entry(char(1024) var, fixed bin(15);

call your_epf(command_args, command_status);
```

command_args is the entire command line as entered by the user.

command_status is the command status returned by the program to operating system. Values returned are:

```
= 0 - no error
> 0 - fatal error
< 0 - soft error or warning
```

There are many examples of programs that can be called commands. In the present version of PRIMOS, commands such as LD, COPY, and DELETE need information from the command line. In fact, these commands are already EPFs and use the EPF calling sequence.

EPF Command Calling Sequence With CPL Local Variables: Commands may be executed from within a CPL file. The CPL file may contain some local variables that are needed by the command. To get these variables, the following interface has been defined.

```
dcl your_epf entry(char(1024) var, fixed bin(15),
                   1, 2 char(32) var,
                      2 fixed bin (15),
                      2 ptr);

call your_epf(command_args, command_status, command_state);
```

command_args is the entire command line as entered by the user.

command_status is the command status returned by the program to the operating system. Values returned are:

```
= 0 - no error
> 0 - hard error
< 0 - soft error or warning
```

command_state is information relative to this invocation. Its contents, in the order specified, are:

command_name	is the command name as entered by the user.
version	is the current version of command state structure; currently this value is = 0.
vcb_ptr	is a 2 word pointer to CPL local variables.

When you write a command that needs to reference CPL local variables, use the vcb_ptr as the argument to be passed to the CPL routines LV$GET and LV$SET to retrieve and set the value of CPL local variables respectively. For example:

```
dcl lv$get entry (ptr options(short), char(32) var,
                  char(1024) var, fixed bin(15), fixed bin(15));

call lv$get (vcb_ptr, var_name, var_value, var_size, err_code);
```

vcb_ptr	is a pointer to CPL local variables.
var_name	is a CPL local variable name.
var_value	is the value of the CPL local variable.
var_size	is the maximum length in chars of the user buffer, var_size.
code	is the standard return error code (0 for success).

```
dcl lv$set entry (ptr options(short), char(32) var,
                  char(1024) var, fixed bin(15));

call lv$set (vcb_ptr, var_name, var_value, err_code);
```

vcb_ptr	is a pointer to CPL local variables.
var_name	is a CPL local variable name.
var_value	is the value of the CPL local variable.
code	is the return error code.

EPF Command Calling Sequence With Command Processor Features: You may find it necessary to determine whether or not the operating system command processor has performed any preprocessing. Preprocessing is automatically done by the command processor whenever certain command line arguments such as '-file' and '-no_verify' are given. The preprocessor options fall into three categories:

- Date restrictions
- Type of objects being used by the command
- Command processing options

To find out if any preprocessing has been done, use the following calling sequence:

```
dcl your_epf entry(char(1024) var, fixed bin(15),
                   1, 2 char(32) var,
                      2 fixed bin (15),
                      2 ptr,
                      2, 3 fixed bin(31),
                         3 fixed bin(31),
                         3 fixed bin(31),
                         3 fixed bin(31),
                         3 bit(1),
                         3 bit(1),
                         3 bit(1),
                         3 bit(1),
                         3 bit(1),
                         3 bit(11),
                         3 bit(1),
                         3 bit(1),
                         3 bit(14),
                         3 fixed bin(15),
                         3 fixed bin(15),
                         3 bit(1),
                         3 bit(1),
                         3 bit(1),
                         3 bit(13);

call your_epf(command_args, command_status, command_state);
```

These arguments are defined as follows:

command_args — The entire command line as entered by user

command_status — The command status returned by the program to the operating system:

= 0 : no error
> 0 : fatal error
< 0 : soft error or warning

command_state — Information relative to this invocation. It contains, in the order specified:

command name — command entered by user

version - current version of the structure of the command state (1 at Rev. 19.4)

vcb_ptr — pointer to CPL local variables

preprocessing_info — information relating to what has been preprocessed:

mod_after_date — if nonzero, then the command processor has found something modified after the given date

mod_before_date — if nonzero, then the command processor has found something modified before the given date

bk_after_date — if nonzero, then the command processor has found something backed up after the given date

bk_before_date — if nonzero, then the command processor has found something backed up before the given date

type_dir — a directory has been found which matches a wildcard

type_segdir — a segment directory has been found that matches a wildcard

type_file — a file has been found that matches a wildcard

type_acat — an access category has been found that matches a wildcard

type_rbf — a ROAM based file has been found that matches a wildcard

mbz1 — 11 bits that are undefined

verify_sw — the -VERIFY option has been given

botup_sw — perform full treewalk before executing program

mbz2 — 14 bits that are undefined

walk_from — the tree level at which the present treewalk started

walk_to — the present treewalk level

in_iteration — command processor is currently in an iteration sequence

in_wildcard — command processor is currently in a wildcard sequence

in_treewalk — command processor is currently in a treewalk sequence

mbz3 — 13 bits that are undefined

To determine if any of the options has been used, you need only check the appropriate field in the 'command_state' parameter. This information is set up at the execution time of the EPF.

EPF Command Function Calling Sequence: You may want to create programs that pass back information to their callers. The information is normally passed directly back to the caller via an assignment statement. An example of this type of program is the string function, LENGTH. LENGTH takes as its argument a target string. It passes back a count of the number of characters in the string. For example:

```
number_of_characters_in_string = length(target_string);
```

EPFs allow this kind of activity at command level. In other words, you can write a program that directly passes some result back to the operating system. This is in addition to the command's status. To do this use the interface below.

```
dcl your_epf entry(char(1024) var, fixed bin(15),
                   1, 2 char(32) var,
                      2 fixed bin (15),
                      2 ptr,
                      2, 3 fixed bin(31),
                         3 fixed bin(31),
                         3 fixed bin(31),
                         3 fixed bin(31),
                         3 bit(1),
                         3 bit(1),
                         3 bit(1),
                         3 bit(1),
                         3 bit(1),
                         3 bit(11),
                         3 bit(1),
                         3 bit(1),
                         3 bit(14),
                         3 fixed bin(15),
                         3 fixed bin(15),
                         3 bit(1),
                         3 bit(1),
                         3 bit(1),
                         3 bit(13);
                   1, 2 bit(1),
                      2 bit(1),
                      2 bit (14),
                   1, 2 bit(1),
                      2 bit(15),
                   ptr);

call your_epf(command_args, command_status, command_state,
              command_fcn_flags, rtn_fcn_ptr);
```

These arguments are defined as follows:

command_args — The entire command line as entered by user

command_status — The command status returned by the program to the operating system:

```
= 0   : no error
> 0   : fatal error
< 0   : soft error or warning
```

command_state — Information relative to this invocation. It contains, in the order specified:

command name — command entered by user

version - current version of the structure of the command state (1 at Rev. 19.4)

vcb_ptr — pointer to CPL local variables

preprocessing_info — information relating to what has been preprocessed:

mod_after_date — if nonzero, then the command processor has found something modified after the given date

mod_before_date — if nonzero, then the command processor has found something modified before the given date

bk_after_date — if nonzero, then the command processor has found something backed up after the given date

bk_before_date — if nonzero, then the command processor has found something backed up before the given date

type_dir — a directory has been found which matches a wildcard

type_segdir — a segment directory has been found that matches a wildcard

type_file — a file has been found that matches a wildcard

type_acat — an access category has been found that matches a wildcard

type_rbf — a ROAM based file has been found that matches a wildcard

mbz1 — 11 bits that are undefined

verify_sw — the -VERIFY option has been given

botup_sw — perform full treewalk before executing program

mbz2 — 14 bits that are undefined

walk_from — the tree level at which the present treewalk started

walk_to — the present treewalk level

in_iteration — command processor is currently in an iteration sequence

in_wildcard — command processor is currently in a wildcard sequence

in_treewalk — command processor is currently in a treewalk sequence

mbz3 — 13 bits that are undefined

command_fcn_flags — information relative to this command function invocation. Its contents in the order specified are:

command_fcn_call — indicates that this program has been called as a command function

no_eval_vbl_fcns — when set, this indicates that command function and global variable references are not to be evaluated

mbz — 14 bits that are undefined

rtn_fcn_ptr — pointer to a structure that describes the values returned to the caller of the EPF function. This structure is itself defined as:

```
dcl 1 rtn_fcn_struc,
        2 version fixed bin(15),
        2 value_str char(*) var;
```

Where:

version — is the structure's version (see ensuing discussion)

value_str — is a string of 1 to 32767 (or any language-imposed limit) characters holding the value to be returned.

First obtain the value of rtn_fcn_ptr by calling ALC$RA (or ALS$RA). After the call to ALC$RA, your program must set the version number of rtn_fcn_struc to 0 and copy the value of that structure into value_str. Then the interface sets rtn_fcn_ptr in its main entrypoint's calling sequence and returns to the calling program. To get the space needed for rtn_fcn_struc use the following:

```
dcl alc$ra entry(fixed bin(31), ptr options(short));

call alc$ra(space_needed, rtn_fcn_ptr);
```

space_needed is the total amount of space needed for rtn_fcn_struc in words. It is the sum of the space needed for your return value and the rtn_fcn_struc version.

Note that ALC$RA provides you with the value for rtn_fcn_ptr which you can then pass back to the command processor.

Library EPFs

Libraries are sets of subroutines that have been bound together into one file. Each subroutine within the file is known an an entrypoint and is available to the PRIMOS dynamic linking mechanism. Therefore, programs external to a library may call subroutines within the library at run time. An example of a library is the FORTRAN library which contains entrypoints for the SINE and COSINE functions.

EPF libraries, like static-mode shared libraries, are linked via the dynamic linking mechanism. This is the main difference between program and library EPFs.

EPF libraries have all of the same properties as EPF programs with the exception of how they are started. Addressing is automatically handled; the library is mapped into memory; its program image may be shared; its linkage is automatically allocated; and so on.

EPF libraries may be created as your own personal libraries. These libraries have the same properties as the standard shared language libraries. The space used for the linkage comes entirely out of your address space. Linkage is generally initialized the first time a link is made to an entrypoint within the library. More about linkage initialization is discussed later.

EPF libraries are not SHARED into memory as are static mode libraries. The normal EPF mechanism is used by the dynamic linking mechanism to bring EPF libraries into memory. This means that an EPF library is not put into your address space until the first time the dynamic linking mechanism needs to look in the library for an entrypoint.

Since EPF libraries are created by you as RUN files, you can restrict access to them by using the file system access control mechanism. That is, you can give certain users valid access rights to your file while denying access rights to others. To take this idea to the extreme, you can create a library that is truly private by only giving yourself access rights to the EPF library.

There are two different types of EPF libraries, Program Class and Process Class. The types are differentiated by when their linkage is initialized. The next three sections discuss the two library classes and the rules involved in linking to libraries.

Program Class Libraries: Program class libraries are directly comparable to the old static mode language libraries. The linkage for these libraries is initialized the first time a link is made to an entrypoint within the library from a new program invocation.

EPF program class libraries are better than static mode libraries in that multiple programs can link to a library simultaneously. Remember, the enhanced command environment allows a program to "call" another program. Both programs may wish to use the same library. If the library were a static mode library, the first time the second program linked to the library, the library would be initialized. Since there is only one linkage area for any static mode library, this initialization would corrupt the linkage with respect to the first program's use of the library. If the library were an EPF program class library, it would be given a new linkage area for every program that linked to it. There would be no corruption of linkage as in the previous example.

Process Class Libraries: EPFs allow you to build libraries which are only initialized once during a login session. These libraries are known as process class libraries. The linkage area for a process class library is allocated only the first time the dynamic linking mechanism looks for an entrypoint within that library.

Process class libraries are useful for sets of entrypoints that only need static data initialized once or that don't need static data at all. The latter case would be a set of entrypoints whose actions are determined solely by their input arguments, constant data, and any local variables (variables that are kept on the stack).

Linking Rules: Since the linkage for the various libraries is initialized at different times, certain kinds of links may not be made. For example, a process class library cannot link to a program class library. This is best shown by a scenario:

Program P1 links to process class library L1

L1 has its linkage initialized

L1 links to program class library L2

L2 has its linkage initialized

L2 returns to L1 which returns to P1

P1 calls program P2

P2 links to process class library L1

L1 links to program class library L2

L2 has its linkage initialized again

This second initialization of L2 may yield different results for L1 than did the first link to L2. L1 would not know this and would generate results based upon the value returned from L2. L1 would then pass this invalid result back to P2. P2 would then generate bad results. The user might never realize anything bad ever happened; not a nice thing to do to a user. Even worse, the programmer would have an even harder time of tracking down this kind of an error.

The following table shows which links are valid:

FROM/TO	PROGRAM	PROGRAM CLASS LIBRARY	PROCESS CLASS LIBRARY	STATIC MODE LIBRARY
PROGRAM	NA	VALID	VALID	VALID
PROG. LIB.	NA	VALID	VALID	VALID
PROC. LIB.	NA	INVALID	VALID	INVALID
STAT. LIB.	NA	VALID	VALID	VALID

In addition, there is one other rule:

> Whenever a static mode library is in use, linking to it is only valid within the same program or library invocation.

Specifically, if program P1 or program class library L1 or static mode library L2 links to static mode library L3, no other program may link to L3 until all program and libraries are fully exited. Fully exited means that the original command level is returned to. Entering a

terminal quit or taking an exception that increases the current command level does not fully exit a program or library. Entering a terminal quit or taking an exception and then releasing to or past the program and/or libraries in question does fully exit the program and/or libraries.

When To Use EPFs

Both program and library EPFs may be used at all times. Their introduction removes the need to build any new static mode programs or libraries. The only exception to this statement is the building of libraries that need to have per-system linkage. Currently, only specially hand-built static mode libraries handle this situation.

How To Build An EPF

EPFs are built in the exact same manner as static mode programs except that you do not have to worry about addressing. You perform all the normal design, editing, and compiling steps. Nothing is different until it is time to load the compiled modules. Instead of using SEG to load your program, you use BIND. If the program requires LOAD, BIND can not handle it; that is, it is probably in R-mode or in S-mode.

What BIND Does

BIND is the new linker for EPFs. It takes the place of SEG; SEG cannot produce EPFs. It is much easier to use than SEG. This is because:

- BIND has only one command that specifies an address. This is the SYMBOL command.
- BIND provides you the ability to tell the system's command processor which of its extended features such as wildcarding or treewalking it should perform.
- BIND can be run either as a subsystem taking its commands one at a time or directly from the command line.
- BIND allows you to build your own personal EPF libraries.
- BIND has a built-in HELP facility.

Using BIND

BIND may either be run from the command line or as a subsystem comparable to SEG. To run BIND from the command line enter:

```
BIND [epf_filename] arguments
```

epf_filename is the name of the file into which BIND places the loaded EPF; if not specified, the EPF takes the name of the first binary loaded

arguments are as follows:

```
-LOAD, -LO list_of_pathnames
-LIBRARY, -LI [list_of_pathnames]
-RELOAD, -RL list_of_pathnames
-ENTRYNAME, -EN list_of_names
-LIBMODE, -LM -PROGRAM | -PROCESS
-SYMBOL, -SY symbol_name definition
```

definition is the <symbol_name> or absolute virtual address <segno>/<wordno>

symbol_name must not be an already defined symbol, or an error results.

For example, to build an EPF named MY_EPF from binaries named MY_PROG1 and MY_PROG2 that need standard libraries functions enter:

```
BIND MY_EPF -LO MY_PROG1 MY_PROG2 -LI
```

This will generate an EPF named MY_EPF.RUN.

To build a program class library named MY_EPFLIB from a binary named MY_ENTRY that contain the entrypoints ENT1, ENT2, and ENT3 enter:

```
BIND MY_EPFLIB -LO MY_ENTRY -LIBMODE -PROGRAM -ENTRYNAME ENT1 ENT2 ENT3
```

This will generate a program class EPF library named MY_EPFLIB.RUN.

To run BIND as a subsytem enter:

```
BIND [epf_filename]
```

All arguments are now entered when the BIND prompt is given in exactly the same manner as commands would be given to SEG. The arguments are as above except that the "-" is not needed. That is, at the BIND prompt, enter LOAD rather than -LOAD. Running BIND in this manner will generate epf_filename.RUN.

How Do I Run An EPF?

Running a program EPF is no different than running any static mode program; just issue the RESUME command for the EPF in question. For example, if you want to run MY_EPF enter:

```
RESUME MY_EPF
```

The operating system will automatically detect that MY_EPF is an EPF rather than an R-Mode image and execute the file as an EPF. The system is able to do this because all EPFs must use the RUN suffix.

Library EPFs are executed in a very different manner. They are dynamically linked to at run time. This happens when a reference is made to an entrypoint within the library. The dynamic linking mechanism automatically detects that the entrypoint is within an EPF library and executes the library as an EPF.

Programming With EPFs

As has been previously mentioned, programming with EPFs is much easier than programming with static mode images because you do not have to worry about addressing. The entire edit, compile, load, execute sequence is faster because the EPF loader, BIND, is much easier to use than either LOAD or SEG. EPFs allow you to write commands much more readily than in the past. Even command functions can now be written.

EPFs give you a choice of writing large applications in two distinct ways; the application may be one large program image or the application may be broken into many different EPF libraries.

The former choice is very much comparable to how applications are built by you today. This can be a very good way to build an application. Using EPFs will make this job go faster because EPFs are faster to develop than are static mode programs.

The latter choice lends itself to programming in a structured environment. You can use EPF libraries to hold all the pieces of your application that you have already debugged. These pieces never have to be reloaded. The current part can be worked on separately as either a program EPF or as another library EPF. You can also structure you application into a set of clearly ordered libraries that can be worked

on separately. Simply put, EPF libraries give you much more flexibility in how you build an application than was ever available before.

The Enhanced Command Environment

One of the main objectives of the EPF product is compatibility. Every program that used to run under PRIMOS must still run under PRIMOS. The command environment should look the same as it always did. To the everyday user, only small changes in the command environment will be evident. Only the more advanced user, and the programmer who makes heavy use of the new features provided with EPFs, will be able to detect all of the new features.

The small changes that the everyday user will see come in two areas; limits on their resource utilization and a slightly higher visibility for command levels. When a users log in, they are placed at command level 1. Whenever an exception is seen or a terminal quit is entered, a new command level is created. The limits on resources are discussed in this chapter. The visibility of command levels is discussed in the section on the Mini-Command Level.

Entrypoint Search List: With the introduction of EPF libraries, especially the ability to define private, per-user libraries, PRIMOS needs a way to find out where the various kinds of libraries may be found. The Entrypoint Search List fulfills this need.

What's A Search List?: A search list is a set of rules that tell the user of the search list where to look for an item. In the context of EPFs, the Entrypoint Search List will tell PRIMOS where in the file system hierarchy to look for a library.

Search List Format: On our system, a search list is an editable ASCII file that contains the rules, one per line, telling the system where to look for the object in question. In other words, to create a search list, all one needs to do is create a file, with any editor, that contains a list of rules, one per line. Comments may also be placed within a search list. Comments must begin with the characters "/*".

A distinction must be made between a search list template file and the current, in-memory version of the search list. A template exists as an editable ASCII file of pathnames. The in-memory version of the search list is updated from the search list template file via a command.

Entrypoint Search List Rules: The Entrypoint Search List may contain three kinds of rules.

- EPF Library Pathname - a pathname to an existing EPF library tells the system to look in that library for the entrypoint.
- Static Mode Library - the string -static_mode_libraries tells the system to scan the set of shared static mode libraries for the entrypoint.
- Use System Default Entrypoint Search List - the string -SYSTEM tells the system to embed the entire system default entrypoint search list at this point in the search list. This search rule is useful onlyf if the -NO_SYSTEM option of the Set_Search_Rule command is used. This command is explained in detail below.

Duplicate rules are not allowed and will be flagged as duplicates.

Most search list rules will be complete EPF pathnames, although full pathnames for library EPFs are not necessary.

The EPF runfile suffix of .RUN may or may not be indicated.

The search list facility will not detect whether or not the EPF library exists.

When the dynamic linking mechanism finds that either a library EPF does not exist or that the user in question does not have the necessary access rights to the library EPF, the search rule which identifies the library will be ignored. No error will be reported. The next search rule in the entrypoint search list will be processed.

System Default Entrypoint List: In order to maintain the basic compatibility of the EPF product, it must be possible for the average user to reach the EPF libraries that are provided with the basic system without having to create their own entrypoint search list. This is accomplished via the System Default Entrypoint List. This list is named ENTRY$.SR and is kept in the directory SYSTEM. It is provided with the master disk software. This file should be ACLd so that it may be altered only by the System Administrator.

Search List Commands: There are two commands that manipulate entrypoint search lists:

```
Set_Search_Rules [ <search_list_pathname> ] [-List_NAMe ENTRY$,
                 [                       ]  -No_System]
                 [ -DeFauLT ENTRY$       ]
                 [ -Help                 ]

List_Search_Rules [ ENTRY$ ]
                  [ -Help  ]
```

The Set_Search_Rules command is used to update the in-memory copy of a user's search list from the file template <search_list_pathname>. The entryname portion of <search_list_pathname> must end in the suffix ENTRY$.SR but may include any other file name components. An exception to this rule is, if the -LIST_NAME ENTRY$ option is used, the entryname portion of <search_list_pathname> does not need to end in the suffix ENTRY$.SR. The .SR suffix does not need to be included on the command line with this command.

If a user does not explicitly set up an ENTRY$ search list, the entrypoint search rules will be set up from the system default file. If a user desires to restore the entrypoint search rules to the system default rules, the -DEFAULT ENTRY$ option to the Set_Search_Rules command should be used.

The default condition is to force the inclusion of the search rules as defined in the system default search list at the head of the user's search list. This will have the effect of forcing the search of the system libraries, as defined in the system default search list SYSTEM>ENTRY$.SR, to be searched before any user private library EPFs. This condition will take effect for those users who either do not change their dynamic linking search list or who issue the Set_Search_Rules command without the -NO_SYSTEM option.

The -NO_SYSTEM option is meant to allow the user to install the search list without having the search rules, as defined in the system default search list, inserted at the head of the search list. If a user issues the SSR command without this option, the system default search rules for that search list will be inserted at the head of the list independent of whether they have included the -SYSTEM search rule as the first search rule in their search list template. If a user issues the SSR command with this option, the user may indicate that the private libraries are to be searched before any of the system libraries. If this option is present on the command line, the system default search rules may be inserted anywhere in their search list by including the -SYSTEM search rule in their template file. If the -NO_SYSTEM option to the SSR command is used and the -SYSTEM search rule is missing, the user will not be able to access any of the system default libraries, unless of course the user specifies the individual library pathnames in the private search list.

Initialization of Dynamic Linking Name Space: Both the user and the system components of a user's dynamic linking name space are initialized each time a process' command environment is initialized: for example, at the issuance of ICE or at process login time or whenever the dynamic linking name space is reset. Therefore, after a process' environment is initialized, its per-process library search list consists entirely of system-wide library entries.

Sample Entrypoint Search Lists: The following sample entrypoint search list is actually an example of a system default entrypoint search list, ENTRY$.SR:

```
/* Unless the -NO_SYSTEM option to the Set_Search_Rules
/* command is used, the system default entrypoint search
/* rules will be inserted here by default.

/* Search my own library EPFs.
    myufd>myepflibs>mylib1.run
    myufd>myepflibs>mylib2.run
    myufd>myepflibs>mylib3.run
```

The following is an example of a tailored entrypoint search list:

```
/* First search my own EPF libraries.
    myufd>myepflibs>mylib1.run
    myufd>myepflibs>mylib2.run
    myufd>myepflibs>mylib3.run
/* Next search all the libraries contained within the
/* system default entrypoint search rules.
    -system
```

Because the user desires to have his own library EPFs searched before any of the system libraries, this search list would need to be set up using the -NO_SYSTEM option to the Set_Search_Rules command.

Interaction With Static Mode Libraries: In order for the system to find any static mode shared library, the -static_mode_libraries rule must be included in either the file SYSTEM>ENTRY$.SR or in your individually defined entrypoint search list.

Programs Calling Programs

One of the best new features of the EPF product allows a currently executing program to actually call another program that is not a library entrypoint without returning to command level. This is done without affecting the calling program's environment. When the called program finishes, execution will continue within the intial program.

When Might A Program Call Another Program: There are basically two different reasons for a program to use this feature:

- To execute an external command as an order from a user. For example, a purchasing system might be created wherein a user is given the commands needed to order various products. When the

user runs the system, a program is executed that reads the user's commands. Each command is kept as an external program in some directory known to the purchasing system program. Whenever the user enters a command, the command reading program calls the appropriate external program to service the user. Since each command is a separate piece, each piece can be altered without affecting any other piece. Also, new commands can be easily added.

- To start another subsystem from within the current subsystem. For example, an engineering development environment subsystem may need to interact with the purchasing system noted above when engineers need to buy parts for their projects. The development environment calls the purchasing system which then asks the engineers for their orders. When all orders have been made, the engineer returns to the development environment.

Limitations On Programs Calling Programs: Since programs may be either RESUMEd or executed as libraries, limitations must be defined for both environments.

Limitations On RESUMEd Programs: Basically, if the programs to be called are EPFs, the only limitation on programs calling programs is defined by the System Administrator on a per-user, per-project, or per-system basis. You can find out what your limit is with the command LIST_LIMITS. These limits are discussed in more detail in the section on New User Attributes. A program can call another program repeatedly and can call as many different programs as it wishes. A program can call another program which can call another program and so on. This all stops as soon as any static mode program is called.

Only one static mode program may be active at any time. Please note that static mode shared libraries are somewhat of an exception to this rule. Static mode programs cannot call static mode programs as the second program would likely overlay the first.

Static mode programs may call EPFs. These EPFs can then call other EPFs. An EPF program may call a static mode program. This static mode program can then call EPF programs.

Limitations On Libraries: Generally the limitations on libraries are the same as those on RESUMEd programs with the addition of the linking rules noted in the section Library EPFs. There is one major exception.

It may be the case that a program called from another program or from a library needs to use a library that is already in use. If this library is not an EPF program class library, then the called program may not use the library since it is already in use. This may be a difficult problem to foresee as the programmer of the caller may not know about the libraries needed by the callee and vice versa.

How A Program Calls Another Program

There are three ways for one program to call another:

- via a simple and friendly interface, CP$
- via a second less friendly interface, EPF$RUN
- via a more complex mechanism which uses the EPF$ routines

For most applications, the CP$ or the EPF$RUN interface will be enough. The first two examples noted above will most likely use the CP$ interface. CP$ can perform all wildcarding, treewalking, and iteration but will not perform abbreviation expansion. If the external program is an EPF and does not need any of these command processor features, EPF$RUN is suitable. Only a few applications need to use the EPF$ routines. Please note that the EPF$ routines can only be used to execute EPFs.

CP$ Interface: The CP$ interface is designed for most applications that need to call external programs. It is designed with ease of use in mind. It permits arguments to be passed to and from the called program. All a programmer has to do is call CP$ with an argument that represents a command line. This pseudo command line will be a character string representation of the external program to be called. The following is the full calling sequence for CP$:

```
dcl cp$ entry (char(1024) var, fixed bin(15), fixed bin(15),
               1, 2 bit(1), 2 bit(14), ptr, ptr);

call cp$ (command_line, status, command_status, command_flags,
          local_variable_ptr, rtn_function_ptr);
```

These arguments have the same meanings as those mentioned previously on the EPF calling sequence. Therefore you need only pass the arguments that your program requires. Remember, you must pass command_line, status, and command_status.

For example, a user may have a purchasing program that allows several different commands, each of which calls an external program that can be called by CP$. The purchasing program prompts the user to insert a command-line; the user inputs ORDER WRENCH (or the longer form below). ORDER is the name of the external program that does the ordering. Part of the purchasing program would therefore resemble the following:

```
/* At this point the User is prompted to input a command. */
/* The User now wants to ORDER WRENCH. But, unless ORDER  */
/* is in CMDNC0, the RESUME command must be added to execute */
/* ORDER, which is probably one of several programs within a  */
/* subdirectory of programs: RESUME PROGS>ORDER WRENCH. */
```

```
/* The subroutine CL$GET is called to gather the terminal input. */

call cl$get(command_line, command_line_length, status);

/* Now CP$ uses that command_line to fetch */
/* the program that will honor this request. */

call cp$('RESUME PROGRS>ORDER WRENCH', status, command_status);
```

Again, remember that CP$ can perform all wildcarding, treewalking, and iteration but will not perform abbreviation expansion.

Using EPF$RUN: The EPF$RUN interface is designed for those applications that need to call external EPF programs. It has also been designed with ease of use in mind. It permits arguments to be passed to and from the called EPF. Basically, all a programmer has to do is open the EPF file and then call EPF$RUN with the arguments needed by the EPF. The following is the full calling sequence for EPF$RUN:

```
dcl epf$run entry (fixed bin(15), fixed bin(15),
                   fixed bin(15), char(1024) var, fixed bin(15),
                   1, 2 char(32) var,
                      2 fixed bin(15),
                      2 ptr,
                      2, 3 fixed bin(31),
                         3 fixed bin(31),
                         3 fixed bin(31),
                         3 fixed bin(31),
                         3 bit(1),
                         3 bit(1),
                         3 bit(1),
                         3 bit(1),
                         3 bit(1),
                         3 bit(11),
                         3 bit(1),
                         3 bit(1),
                         3 bit(14),
                         3 fixed bin(15),
                         3 fixed bin(15),
                         3 bit(1),
                         3 bit(1),
                         3 bit(1),
                         3 bit(13),
                   1, 2 bit(1),
                      2 bit(15),
                   ptr);

epf_id = epf$run (key, unit, status, command_args,
                  command_status, command_state,
                  command_flags, rtn_function_ptr);
```

key is used to tell EPF$RUN how far to go with the execution of the EPF. It has three values:

k$invk	run, and encache, upon completion, the EPF; this is the key you will normally use
k$invk_del	run and delete upon completion but do not encache the EPF; you may use this key upon occasion
k$restore_only	only map, allocate linkage for, and initialize the EPF but do not invoke it; you will probably never need this key

unit is the file on which the EPF is open for VMFA read.

The rest of the arguments have the same meanings as those noted in the previous sections on the EPF calling sequence. Therefore, you need only to pass the arguments that your program requires. Remember you must pass command_line, status, and command_status.

If the EPF you wanted to run was named MY_EPF and MY_EPF needed no arguments, you would use the following code sequence:

```
/* First open the EPF. */

call srsfx$(k$vmr, 'MY_EPF', src_unit, type, n_suffixes,
            suffix_list, basename, suffix_used, status);

/* Now run the EPF. */

epf_id = epf$run (k$invk, unit, status, command_args,
                  command_status, command_state);
```

Mini-Command Level

As has already been mentioned, the EPF product provides the user with a much better command environment in which to build and run programs. Another improvement has also been made which helps the system control the allocation of memory, a very valuable resource.

Whenever any program is run, the program needs memory in which to execute. This memory comes from the system and is shared among all users. As the EPF product allows more than one EPF to be suspended at one time, it is possible for a user to use up a lot of memory without even noticing. Remember, a program is suspended every time you enter a quit or whenever a program exception, such as an access violation, is taken. In addition, EPF libraries use up valuable space in a way that was previously impossible. Therefore, to remind users that they are using a lot of memory, only a fixed number of suspended programs are allowed. (This number is set by the System Administrator and may vary

from user to user. Please see the section on New User Attributes for further information.) When this number is exceeded, the user enters the Mini-Command level.

The Mini-Command level is just like a normal command level except that a fixed subset of PRIMOS commands are allowed. These commands allow users to free up the memory in use by the suspended programs. They prevent any more memory from being used. In addition, private command level abbreviations are disabled at the mini-command level.

When you exceed your allocated number of command levels the following message is printed:

```
You have exceeded your maximum number of command levels.
```

You are now at mini-command level. Only the commands shown below are available. Of these, RLS -ALL should return you to command level 1. If it does not, type ICE. If this problem recurs, contact your System Administrator.

Valid mini-commands are:

Abbrev	Full name	Abbrev	Full name
C	CLOSE	COMO	COMOUT
DMSTK	DUMP_STACK		
ICE	INITIALIZE_COMMAND_ENVIRONMENT	LE	LIST_EPF
LL	LIST_LIMITS	LMC	LIST_MINI_COMMANDS
LS	LIST_SEGMENT	LOGIN	LOGIN
LO	LOGOUT	P	PM
PR	PRERR	RDY	RDY
REN	REENTER	RLS	RELEASE_LEVEL
REMEPF	REMOVE_EPF	S	START

If you try to enter another quit, the following message is printed:

```
Terminal QUIT invalid now.  (listen_)
```

The subset of PRIMOS commands that are permitted may be obtained by entering List_Mini_Commands (LMC) as follows:

```
OK, LIST_MINI_COMMANDS
Abbrev   Full name                              Abbrev   Full name
-------------------------------------------------------------------

C        CLOSE                                  COMO     COMOUT
DMSTK    DUMP_STACK
ICE      INITIALIZE_COMMAND_ENVIRONMENT         LE       LIST_EPF
LL       LIST_LIMITS                            LMC      LIST_MINI_COMMANDS
LS       LIST_SEGMENT                           LOGIN    LOGIN
LO       LOGOUT                                 P        PM
PR       PRERR                                  RDY      RDY
REN      REENTER                                RLS      RELEASE_LEVEL
REMEPF   REMOVE_EPF                             S        START

OK,
```

At this point, the commands that you will use most often are RELEASE_LEVEL and CLOSE. You should probably use both as follows:

```
C -ALL;RLS -ALL
```

This frees all of the memory that you have used for program EPFs and closes all open file units. Library EPFs remain mapped to your user process.

If you want to return to your initial attach point and run your LOGIN program when you reach the Mini-Command level, just enter the INITIALIZE_COMMAND_ENVIRONMENT (ICE) command.

If your program became suspended because of an exception, such as access violation, you may wish to try diagnosing the problem before releasing the program. Commands such as PM, DMSTK, and COMO are useful for this kind of debugging.

If your program became suspended because you ran out of memory, you may wish to try returning some memory that is not currently in use to the system and then REENTER your program. LIST_EPF (LE) and LIST_SEGMENT (LS) allow you to see what EPF is in which segment and which segments are currently in use. With this information available, you can then use the REMOVE_EPF (REMEPF) command to delete an EPF which will free up its associated memory. You can then try a REENTER (REN) command.

New Commands

Many new commands have been added to the system for use with EPFs. Some of these commands relate to entrypoint search lists, some relate to the enhanced command environment, and some relate directly to EPFs. In addition, the COPY command has been modified to allow copying an in-use EPF.

Entrypoint Search List Commands

Set_Search_Rules:

```
SET_SEARCH_RULES [ search_list_pathname ] [-List_NAMe ENTRY$,
                 [                      ]  -No_System]
                 [ -DeFauLT ENTRY$      ]
                 [ -Help                ]
```

Abbreviation: SSR

This command updates the in-memory copy of a user's search list from the file template <search_list_pathname>.

search_list_pathname	Name of an entrypoint search list. The search list file template name must end in the suffix ENTRY$.SR but may include any other file name components. The .SR suffix does not need to be included on the command line with this command.

For example:

```
SET_SEARCH_RULES MY_UFD>ENTRY$.SR
```

List_Search_Rules:

```
List_Search_Rules [ ENTRY$ ]
                  [ -Help  ]
```

Abbreviation: LSR

This command lists the in-memory copy of a user's search list. For example:

```
LIST_SEARCH_RULES

Search list template file: <DCH1>SYSTEM>ENTRY$.SR

    LIBRARIES*>SYSTEM_LIBRARY.RUN
    LIBRARIES*>FORTRAN_IO_LIBRARY.RUN
    -STATIC_MODE_LIBRARIES
```

Command Environment Commands:

LIST_LIMITS
LL

You use this command to display information on various attributes affecting your command environment. The LIST_LIMITS command displays the following attributes:

- The number of command levels you can use
- The number of programs you can invoke at any command level
- The number of private dynamic segments you can use
- The number of private static segments you can use

LIST_LIMITS is useful when you think you may have exceeded one of these limits. For example, if you have used up all the command levels allocated to you and have reached mini-command level, you can use this command to check your limit.

The System Administrator sets all these attributes either in your user profile or on a system-wide basis.

The following example shows how LIST_LIMITS displays this information.

```
OK, LIST_LIMITS

Maximum number of command levels: 10
Maximum number of program invokations: 10
Maximum number of private static segments: 100
Maximum number of private dynamic segments: 150

OK,
```

INITIALIZE_COMMAND_ENVIRONMENT:

Initialize_Command_Environment
ICE

This command re-initialize the user's Ring 3 command environment and will close all open files including any open COMOUTPUT file, reset the Ring 3 command environment to an initial state, and execute a user's login file.

LIST_SEGMENT:

```
LIST_SEGMENT [ segno-1 ... segno-8 ] [ -STATIC  -DYNAMIC  -NAME   ]
                                     [ -ST      -DY                ]
                                     [ -BRIEF   -NO_WAIT  -HELP    ]
                                     [ -BR      -NW       -H       ]
```

Abbreviation: LS

The LIST_SEGMENT command displays information about the private segments of the current user which are in use. The command displays only the DTAR2 private segments. The DTAR2 segments are defined within a range from '4000 to '5777.

Segment numbers are displayed in ascending numerical order. If you give the command with no options, then LIST_SEGMENT displays only the segment number and the access rights assigned to each segment. Three possible combinations of access rights may be assigned:

Access Code	Access Allowed
Null	No access allowed
RX	Read and execute access
RWX	Read, write, and execute access

The following example shows the output of a LIST_SEGMENT command without options.

```
OK, LIST_SEGMENT

3 Private static segments.
segment access
---------------
4000    RWX
4001    RWX
4002    RWX

4 Private dynamic segments.
segment access
---------------
4340    RX
4341    RX
4375    RWX
4377    RWX
```

You use segno-1 ... segno-8 to specify up to eight octal segment numbers on which you want information. You cannot include wildcards in the segment numbers or use iteration with them. If you do not give any segment numbers, LIST_SEGMENT displays information for each segment that is currently in use in the static and dynamic segment ranges.

The following examples show two uses of LIST_SEGMENT with specified segment numbers.

```
OK, LIST_SEGMENT 4000 4001 4005 4377

2 Private static segments.
segment access
-------------
4000    RWX
4001    RWX

1 Dynamic Segments.
segment access
-------------
4377    RWX

Private static segment 4005 is not currently in use.

OK, LIST_SEGMENT 4374 4376 4372

3 specified private dynamic segments are not currently in use.
```

Option	Meaning
[-STATIC] [-ST]	Displays information only about static private segments.
[-DYNAMIC] [-DY]	Displays information only about dynamic private segments.
[-NAME]	Displays the name of any EPF file which is associated with the segment. (An EPF may be associated with a segment if the procedure or the linkage areas for that EPF are assigned to that segment.) This option is valid only for your own private, dynamic segments. If more than one EPF is associated with a given segment, as may happen if the linkage areas of several EPFs are allocated within the same segment, the EPF pathnames are displayed alphabetically by filename, one per line.

If a given EPF uses more than one segment, the EPF pathname will appear alongside the segment number/access right pair for each segment.

If a dynamic segment is not associated with an EPF, the word "none" appears by that segment.

[-BRIEF]
[-BR]

Displays only total number of segments which are currently in use in each segment range.

The following example shows the use of the -NAME option.

```
OK, LIST_SEGMENT -NAME

3 Private static segments.
segment access
-------------
4000    RWX
4001    RWX
4002    RWX

4 Private dynamic segments.
segment access  epf
-----------------
4340    RX      <DEN>LIBRARIES*>SYSTEM_LIBRARY.RUN
4341    RX      <DEN>LIBRARIES*>FORTRAN_LIBRARY.RUN
4375    RWX     (none)
4377    RWX     <DEN>LIBRARIES*>FORTRAN_LIBRARY.RUN
                <DEN>LIBRARIES*>SYSTEM_LIBRARY.RUN
```

The next example shows the summary display provided by the -BRIEF option.

```
OK, LIST_SEGMENT -BRIEF

3 Private static segments.

4 Private dynamic segments.
```

To control screen scrolling, use the -NO_WAIT option. To remind yourself of the syntax of the command, use the -HELP option.

Option	Meaning
[-NO_WAIT] [-NW]	Enables terminal screen scrolling. This option suppresses the —More— prompt that is otherwise given at the end of each 23 lines of display. If you do not specify -NO_WAIT, PRIMOS prompts you before scrolling the terminal screen. To display the next screen of output, respond Y, YES, OK, NEXT, or press RETURN. To exit from the command, respond N, NO, Q, or QUIT.
[-HELP] [-H]	Displays the syntax of LIST_SEGMENT. The -HELP display is also printed if PRIMOS encounters an error while parsing the command.

List_Mini_Commands:

LIST_MINI_COMMANDS [command_match]

Abbreviation: LMC

LIST_MINI_COMMANDS displays the names of PRIMOS commands that you can use after you reach mini-command level, as explained earlier.

A command_match is possibly a wildcard-laden character string that is used as a pattern match for mini-commands to be listed. If you leave out this argument, LIST_MINI_COMMANDS displays the names of all the PRIMOS commands that you can use at mini-command level.

For example, if you do not specify a command match:

```
OK, LIST_MINI_COMMANDS
Abbrev   Full name                              Abbrev   Full name
-------------------------------------------------------------------

C        CLOSE                                  COMO     COMOUT
DMSTK    DUMP_STACK
ICE      INITIALIZE_COMMAND_ENVIRONMENT         LE       LIST_EPF
LL       LIST_LIMITS                            LMC      LIST_MINI_COMMAND
S
LS       LIST_SEGMENT                           LOGIN    LOGIN
LO       LOGOUT                                 P        PM
PR       PRERR                                  RDY      RDY
REN      REENTER                                RLS      RELEASE_LEVEL
REMEPF   REMOVE_EPF                             S        START
```

If you specify the command match LIST@@:

```
OK, LIST_MINI_COMMANDS LIST@@
Abbrev   Full name                          Abbrev   Full name
-------------------------------------------------------------

LE       LIST_EPF                           LL       LIST_LIMITS
LMC      LIST_MINI_COMMANDS                 LS       LIST_SEGMENT

OK,
```

EPF Commands:

```
LIST_EPF [ pathname-1 ... pathname-8 ]

        [ -ACTIVE  -NOT_ACTIVE  -NOT_MAPPED ]
        [ -AC      -NA          -NM         ]

        [ -PROGRAM  -LIBRARY  -SEGMENTS ]
        [ -PRG      -LI       -SEGS     ]

        [ -COMMAND_PROCESSING ]
        [ -CP                 ]

        [ -EPF_DATA ]
        [ -ED       ]

        [ -DETAIL ]
        [ -DET    ]

        [ -NO_WAIT  -HELP ]
        [ -NW       -H    ]
```

Abbreviation: LE

You use this command to display information on EPFs. The LIST_EPF command can display information about EPFs whether or not the EPF is currently mapped to your address space. That is, the command works on two domains:

1. Your address space. You can display information on any or all of the EPFs that are mapped into your address space. For example, if you quit from a running EPF, it remains mapped into your space.

2. The file system. You can display information on any EPF by giving its pathname. If you want to find out about an EPF that's not mapped to your address space, use the -NOT_MAPPED option, described below.

Unless you use the -NOT_MAPPED option, LIST_EPF looks for an EPF that is already mapped into your address space. If you give the pathname of an EPF that is not mapped, without specifying the -NOT_MAPPED option, LIST_EPF displays no information about that EPF.

You can use pathname-1 ... pathname-8 to specify up to eight pathnames of EPFs. You need not include the EPF suffixes .RUN or .RPn, (where n is one of 10 decimal digits). These pathnames may be simple filenames or full pathnames.

You may include wildcards within the entryname portion (that is, the final component) of the pathname but LIST_EPF does not support treewalking or iteration.

This example shows how to use LIST_EPF with a pathname:

```
OK, LIST_EPF <MAGNUM>LIBRARIES*>@@

2 Process-Class Library EPFs.

(not_active) <MAGNUM>LIBRARIES*>FORTRAN_LIBRARY.RUN
(active)     <MAGNUM>LIBRARIES*>SYSTEM_LIBRARY.RUN

1 Program-Class Library EPF.

(not_active) <MAGNUM>LIBRARIES*>FORTRAN_IO_LIBRARY.RUN

OK,
```

If you give the command with no options, LIST_EPF displays the full pathname of the EPF file or files, and sorts the files by type. This type may be one of:

- Process-Class Library EPF
- Program-Class Library EPF
- Program EPF

Within these types, LIST_EPF displays the names in alphabetical order based on the filename of the EPF. The names of the EPFs displayed include the .RUN or any of the .RPn suffixes, whichever applies.

LIST_EPF also displays the status of each EPF. The status of an EPF may be one of these three: active, not active, or not mapped.

Active: PRIMOS treats an EPF as active if it is mapped to a user's address space and:

1. The EPF is a program or program-class library EPF that has been suspended while executing or

2. The EPF is a process-class library and has been initialized. This is also called an in-use EPF.

Not active: PRIMOS treats an EPF as not active, or inactive, if it is mapped to the user's address space but is neither a suspended EPF nor an in-use process-class library EPF.

Not mapped: PRIMOS classifies all other EPFs as not mapped.

If you use LIST_EPF with a filename, rather than a pathname, the command displays the full pathnames of all the EPFs with that filename, as this example shows:

```
OK, LIST_EPF LD

2 Program EPFs.

(active)      <DEN>CMDNC0>LD.RUN
(active)      <DEN>TIM>LD.RUN
```

If you do not specify a pathname, LIST_EPF displays information for all EPFs currently mapped to your address space, as shown in the following example:

```
OK, LIST_EPF

1 Program-Class Library EPF.

(not active) <DEN>LIBRARIES*>FORTRAN_IO_LIBRARY.RUN

1 Process-Class Library EPF.

(active)      <DEN>LIBRARIES*>SYSTEM_LIBRARY.RUN

3 Program EPFs.

(active)      <DEN>CMDNC0>LD.RUN
(active)      <DEN>TIM>LD.RUN
(not active) <DEN>TIM>SUB1>SUB2>SUB3>SUB4>SUB5>SUB6>MY_PROGRAM.RUN
```

If the file you specify in pathname does not exist, you see the message shown in the following example:

```
OK, LIST_EPF SPENSER>FAERIE_QUEEN

No entries selected.

OK,
```

To select the kind of EPF to be listed, by type or status, you can use the following options:

Option	Meaning
[-ACTIVE] [-AC]	Selects only active EPFs.
[-NOT_ACTIVE] [-NA]	Selects only non-active EPFs.
[-NOT_MAPPED] [-NM]	Displays information for the EPF file given by pathname. If no pathname is specified, displays information for all EPF files in the user's current (working) directory.
[-PROGRAM] [-PRG]	Selects only program EPFs.
[-LIBRARY] [-LI]	Selects only library EPFs.

The following examples show the use of the -ACTIVE and -NOT_ACTIVE options.

```
OK, LIST_EPF -ACTIVE

1 Process-Class Library EPF.

(active)      <DEN>LIBRARIES*>SYSTEM_LIBRARY.RUN

2 Program EPFs.

(active)      <DEN>CMDNC0>LD.RUN
(active)      <DEN>TIM>LD.RUN


OK, LIST_EPF -NOT_ACTIVE

1 Program-Class Library EPF.

(not active) <DEN>LIBRARIES*>FORTRAN_IO_LIBRARY.RUN

1 Program EPF.

(not active) <DEN>TIM>SUB1>SUB2>SUB3>SUB4>SUB5>MY_PROGRAM.RUN
```

To find out what segments and linkage areas your EPFs are using, specify the -SEGMENTS option.

Option	Meaning
[-SEGMENTS] [-SEGS]	For all EPFs that are currently mapped into your address space, displays: 1. The type of the EPF 2. The status of the EPF 3. The full pathname of the EPF 4. The number of procedure segments being used by the EPF 5. For each procedure segment in use, two numbers separated by a colon. The number to the left of the colon is an even integer greater than or equal to zero, preceded by a + sign. The number to the right of the colon shows the actual segment number used for the EPF procedure. The integer on the left relates the actual segment number to the imaginary segment number indicated by the same even integer in the BIND map for the EPF. 6. The number of linkage areas being used by the EPF 7. For each linkage area in use, two numbers separated by a colon. The number to the left of the colon is an even integer less than zero, preceded by a - sign. The number to the right of the colon shows the segment/word number pair of the linkage area used for the most recent invocation of the EPF. The integer on the left relates the actual segment number to the imaginary segment number indicated by the same integer in the BIND map for the EPF. If procedure segments or linkage areas have not yet been allocated to the EPF, the phrase "not allocated" is displayed.

The following examples show the use of the -SEGMENTS option, both without and with a pathname.

```
OK, LIST_EPF -SEGMENTS

1 Process-Class Library EPF.

(active)      <DEN>LIBRARIES*>SYSTEM_LIBRARY.RUN
   7 procedure segments:    +0: 4370          +2: 4373
                            +4: 4374          +6: 4375
                           +10: 4376         +12: 4377
                           +14: 4402
   3 linkage areas:         -2: 4365/55342    -4: 4366/0
                            -6: 4367/176503

1 Program-Class Library EPF.

(not active) <DEN>LIBRARIES*>FORTRAN_IO_LIBRARY.RUN
   2 procedure segments:    +0: 4320          +2: 4322
   1 linkage area:          -2: 4377/25637

3 Program EPFs.

(active)      <DEN>CMDNC0>LD.RUN
   1 procedure segment:     +0: 4370
   1 linkage area:              (not allocated)
(active)      <DEN>TIM>LD.RUN
   1 procedure segment:     +0: 4045
   5 linkage areas:         -2: 4046/176500   -4: 4047/5251
                            -4: 4047/40454   -10: 4050/52510
                           -12: 4052/0
(not active) <DEN>TIM>SUB1>SUB2>SUB3>SUB4>SUB5>MY_PROGRAM.RUN
   3 procedure segments:    +0: 4015          +2: 4020
                            +4: 4021
   2 linkage areas:             (not allocated)

OK, LIST_EPF COPY -SEGMENTS

No entries selected.
```

The following example shows the use of the -SEGMENTS and the -NOT_MAPPED options together, for a user attached to the directory CMDNC0.

```
OK, LIST_EPF COPY -NOT_MAPPED -SEGMENTS

1 Program EPF.

(not mapped) <DEN>CMDNC0>COPY.RUN
   2 procedure segments:  (not allocated)
   2 linkage areas:       (not allocated)
```

To look at the state of command processing features for a program EPF, use the -COMMAND_PROCESSING option.

Option	Meaning
[-COMMAND_PROCESSING] [-CP]	Displays the full pathname of the EPF and, for a program EPF, examines command processing features, such as: 1. The type of file system objects on which the EPF may operate. 2. Whether the command processor should process wildcarding, treewalking, or command iteration for the EPF concerned. 3. The name generation position for the EPF. For the first two categories, the presence of the terms indicates that the feature is enabled. Command processing information is not relevant for library EPF, and is not displayed.

For example:

```
OK, LIST_EPF -COMMAND_PROCESSING

1 Process-Class Library EPF.

(active)      <DEN>LIBRARIES*>SYSTEM_LIBRARY.RUN

1 Program-Class Library EPF.

(not active) <DEN>LIBRARIES*>FORTRAN_IO_LIBRARY.RUN

3 Program EPFs.

(active)      <DEN>CMDNC0>LD.RUN
   command options: wldcrd,trwlk,iter  file,dir,segdir,acat,  1
(active)      <DEN>TIM>LD.RUN
   command options: wldcrd,trwlk,iter  file,dir,segdir,acat,  1
(not active) <DEN>TIM>SUB1>SUB2>SUB3>SUB4>SUB5>MY_PROGRAM.RUN
   command options:  (none)              (none)                5
```

Option	Meaning
[-EPF_DATA -ED]	For the specified EPF or EPFs, displays the following information: 1. The type, status, and full pathname of the EPF 2. The version of BIND used to create the EPF 3. The date on which the EPF was bound 4. The program name of the EPF 5. The user version of the EPF 6. The contents of the EPF comment field 7. The number of debugger segments being used by the EPF If the EPF was bound by a version of BIND which cannot supply data on items 2 through 6, the following message is displayed instead: EPF data not available. For example, some PRIMOS commands were created as EPFs prior to Rev. 19.4: the BIND used to create them may not supply this information.

For example:

```
OK, LIST_EPF -EPF_DATA

1 Process-Class Library EPF.

(active)      <DEN>LIBRARIES*>SYSTEM_LIBRARY.RUN
   bind version:    19.4.1
   date of binding: 01 Jan 84 00:00:01
   program name:    system_library
   user version:    2.7.my_release
   comment:         This library is guaranteed to be bug-free.
   debug segments:  (none)

1 Program-Class Library EPF.

(not active) <DEN>LIBRARIES*>FORTRAN_IO_LIBRARY.RUN
   bind version:    19.4.1
   date of binding: 12 Oct 83 16:25:56
   program name:    fortran_io_library
   user version:    2.7.my_release
   comment:         Copyright (c) 1983, Prime Computer, Inc.,
                    Natick, MA 01760 All Rights Reserved
   debug segments:  5

3 Program EPFs.

(active)      <DEN>CMDNC0>LD.RUN
   bind version:    19.4.1
   date of binding: 09 Sep 83 11:23:59
   program name:    ld
   user version:    (none)
   comment:         (none)
   debug segments:  (none)

(active)      <DEN>TIM>LD.RUN

   bind version:    19.4.1
   date of binding: 05 May 83 05:05:05
   program name:    ld
   user version:    20.0.my_release
   comment:         This EPF had better fix the bug in the
                    installed version of LD.RUN.
   debug segments:  (none)

(not active) <DEN>TIM>SUB1>SUB2>SUB3>SUB4>SUB5>SUB6>MY_PROGRAM.RUN

   bind version:    19.4.1
   date of binding: 03 Mar 83 00:00:09
   program name:    ld
   user version:    20.0.1
   comment:         No specific comment to place here now.
   debug segments:  (none)
```

Option	Meaning
[-DETAIL] [-DET]	Displays all attributes for each entry selected. These attributes include the ones displayed by the options -COMMAND_PROCESSING, -EPF_DATA and -SEGMENTS.

For example:

```
OK, LIST_EPF -DETAIL

1 Process-Class Library EPF.

(active)      <DEN>LIBRARIES*>SYSTEM_LIBRARY.RUN
   7 procedure segments:    +0: 4370          +2: 4373
                            +4: 4374          +6: 4375
                           +10: 4376         +12: 4377
                           +14: 4402
   3 linkage areas:         -2: 4365/55342    -4: 4366/0
                            -6: 4367/176503
   bind version:    19.4.1
   date of binding: 01 Jan 84 00:00:01
   program name:    system_library
   user version:    2.7.my_release
   comment:         This library is guaranteed to be bug-free.
   debug segments:  (none)

1 Program-Class Library EPF.

(not active) <DEN>LIBRARIES*>FORTRAN_IO_LIBRARY.RUN
   2 procedure segments:    +0: 4320          +2: 4322
   1 linkage area:          -2: 4377/25637
   bind version:    19.4.1
   date of binding: 12 Oct 83 16:25:56
   program name:    fortran_io_library
   user version:    2.7.my_release
   comment:         Copyright (c) 1983, Prime Computer, Inc.,
                    Natick, MA 01760 All Rights Reserved
   debug segments:  5

3 Program EPFs.

(active)      <DEN>CMDNCO>LD.RUN
   1 procedure segment:     +0: 4370
   1 linkage area:              (not allocated)
   bind version:    19.4.1
   date of binding: 09 Sep 83 11:23:59
   program name:    ld
   user version:    (none)
   comment:         (none)
   debug segments:  (none)
   command options: wldcrd,trwlk,iter   file,dir,segdir,acat,rbf  1
```

```
(active)      <DEN>TIM>LD.RUN
   1 procedure segment:     +0: 4045
   5 linkage areas:         -2: 4046/176500    -4: 4047/5251
                            -6: 4047/40454    -10: 4050/52510
                           -12: 4052/0
   bind version:    19.4.1
   date of binding: 09 Sep 83 11:23:59
   program name:    ld
   user version:    20.0.my_release
   comment:         This EPF had better fix the bug in the
                    installed version of LD.RUN.
   debug segments:  1
   command options: wldcrd,trwlk,iter   file,dir,segdir,acat,rbf  1

(not active) <DEN>TIM>SUB1>SUB2>SUB3>SUB4>SUB5>SUB6>MY_PROGRAM.RUN
   3 procedure segments:    +0: 4015          +2: 4020
                            +4: 4021
   2 linkage areas:             (not allocated)
   bind version:    19.4.1
   date of binding: 03 Mar 83 00:00:09
   program name:    ld
   user version:    20.0.1
   comment:         I do not really have a specific comment to
                     place here now.
   debug segments:  29
   command options: (none)              (none)                 1
```

To control screen scrolling, use the -NO_WAIT option. To remind yourself of the syntax of the command, use the -HELP option.

Option	Meaning
[-NO_WAIT -NW]	Enables terminal screen scrolling. This option suppresses the —More— prompt that is otherwise given at the end of each 23 lines of display. If you do not specify -NO_WAIT, PRIMOS prompts you before scrolling the terminal screen. To display the next screen of output, respond Y, YES, OK, NEXT, or press RETURN. To exit from the command, respond N, NO, Q or QUIT.
[-HELP -H]	Displays the syntax of LIST_EPF. This HELP display is also printed if PRIMOS encounters an error while parsing the command.

REMOVE_EPF:

```
REMOVE_EPF  [ pathname ] [ -ACTIVE  -NOT_ACTIVE ]
                         [ -AC      -NA         ]

                         [ -VERIFY  -NO_VERIFY  ]
                         [ -VFY     -NVFY       ]

                         [ -QUERY   -NO_QUERY   ]
                         [          -NQ         ]

                         [ -HELP    ]
                         [ -H       ]
```

Abbreviation: REMEPF

You use this command to remove an EPF from your address space. That is, if an EPF is mapped to your address space, REMOVE_EPF will unmap it. REMOVE_EPF does not remove suspended EPFs.

REMOVE_EPF does not delete the EPF file itself. The command is useful if:

- You want to delete an EPF that is currently mapped to your address space. Before you can use the DELETE command, you have to remove the EPF. If you try to delete the EPF without first removing it, you will get the error message 'File in use'.

- While you have been using one version of an EPF, another user has replaced it (using the REPLACE function of COPY or BIND). If you want to use the new version, you must first remove the old one.

- If you want to use a new version of an EPF, but have the old one mapped into your address space. You do not always have to use REMOVE_EPF in this case: for example, if you rebind an EPF, BIND automatically removes the previous version. Removal is also automatic if you are using the debugger.

You can specify the pathname of the EPF file you want to remove. You can use a simple filename or a full pathname. You do not have to include the EPF .RUN or .RPn suffixes. Because REMOVE_EPF supports command processor iteration, the pathname can include wildcards. The command does not, however, support treewalking.

If you do not give a pathname, REMOVE_EPF assumes that you want to remove all the non-suspended EPFs in your address space. The command asks you which EPFs you wish to remove, as shown in the following example. To remove the EPF, answer Y or YES. To leave the EPF alone, answer N or NO.

```
OK, REMOVE_EPF
Ok to remove EPF file <OSGRP0>LIBRARIES*>FORTRAN_IO_LIBRARY.RUN? NO
Ok to remove EPF file <OSGRP0>LIBRARIES*>FORTRAN_LIBRARY.RUN? NO
Ok to remove EPF file <OSGRP0>LIBRARIES*>SYSTEM_LIBRARY.RUN? NO

No EPFs removed (REMOVE_EPF).
OK,
```

If the EPF file you specify does not exist, or is not already mapped into your address space, you will get the message shown in this example:

```
OK, REMOVE_EPF MISTAKE

No EPFs removed (REMOVE_EPF).
OK,
```

The options for REMOVE_EPF are:

Option	Meaning
[-ACTIVE] [-AC]	Terminates only in-use process-class library EPFs. You cannot use REMOVE_EPF to remove suspended EPFs.
[-NOT_ACTIVE] [-NA]	Terminates only non-active EPFs. These are EPFs which are currently mapped to the user's address space but which are neither suspended EPFs nor in-use process-class library EPFs.
[-VERIFY] [-VFY]	Requests the user to verify all EPF terminations. By default, you are asked to verify terminations only when you include wildcards in pathname.
[-NO_VERIFY] [-NVFY]	Suppresses verification checking when you include wildcards in the pathname. You cannot use the -VERIFY and -NO_VERIFY options together.
[-QUERY]	Requests the user to verify that an EPF is to be removed if the EPF is currently in use within the user's address space. This is the default.
[-NO_QUERY] [-NQ]	Suppresses user verification if the EPF is currently in use within the user's address space. You cannot use the -QUERY and -NO_QUERY options together.

[-HELP -H]	Displays the syntax of REMOVE_EPF. The help display is also printed if PRIMOS encounters an error while parsing the command.

In the following example, the user MAGGIE lists her EPFs, removes a FORTRAN library EPF, and finally removes all her inactive EPFs.

```
OK, LIST_EPF

1 Process-Class Library EPF.

(active)      <DEN>LIBRARIES*>SYSTEM_LIBRARY.RUN

1 Program-Class Library EPF.

(not active) <DEN>LIBRARIES*>FORTRAN_IO_LIBRARY.RUN

3 Program EPFs.

(active)      <DEN>CMDNC0>LD.RUN
(active)      <DEN>MAGGIE>LD.RUN
(not active) <DEN>MAGGIE>SUB1>SUB2>SUB3>SUB4>SUB5>
             MY_PROGRAM.RUN

OK, REMOVE_EPF FORTRAN@@
Ok to remove EPF <DEN>LIBRARIES*>FORTRAN_IO_LIBRARY.RUN? YES


OK, REMOVE_EPF @@ -NOT_ACTIVE
Ok to remove EPF <DEN>LIBRARIES*>FORTRAN_IO_LIBRARY.RUN? YES
Ok to remove EPF <DEN>MAGGIE>SUB1>SUB2>SUB3>SUB4>SUB5>
                 MY_PROGRAM.RUN? YES
```

List_Library_Entries:

```
LIST_LIBRARY_ENTRIES [ pathname-1 ... pathname-8 ]

                     [ -ENTRYNAME entryname-1 ... entryname-8 ]
                     [ -EN                                    ]

                     [ -NO_WAIT  -HELP ]
                     [ -NW       -H    ]
```

Abbreviation: LLENT

This command displays alphabetically-sorted selected entrypoints in a library EPF where pathname identifies the library EPF. You can use the optional entrynames to select the library entrypoints to display. The entrynames can include wildcards.

You can use pathname-1 ... pathname-8 to specify up to eight pathnames of library EPFs. Wildcarding is supported only within the entryname portion of the pathnames, which means that treewalking is not supported. You do not have to include the suffixes .RUN or .RPn.

If you do not specify a pathname, information is displayed for all EPFs currently mapped to your address space.

The options for LIST_LIBRARY_ENTRIES are:

Option	Meaning
[-ENTRYNAME] [-EN] entryname-1 ... entryname-8	Displays selected entrypoints within a library EPF. You use the entrynames, which may include wildcards, to specify which entrypoints you want displayed. If you do not specify entrynames, or do not use the -ENTRYNAME option, all of the entrypoints within the library EPF are displayed.

By default, LIST_LIBRARY_ENTRIES displays entrypoints in seven columns per line of display. If the name of an entrypoint runs into an adjacent column, fewer than seven names are displayed on the affected line. For example:

```
OK, LIST_LIBRARY_ENTRIES LIBRARIES*>FORTRAN_IO_LIBRARY -EN F$@@

(not active)    <DEN>LIBRARIES*>FORTRAN_IO_LIBRARY.RUN.

Program-Class Library EPF, 104 Total Entrypoints, 12 Selected Entrypoin
ts.

F$ABCDEFG               F$BCDEFG                F$CDEFG                 F$DEFG
F$E                     F$F                     F$GHIJKLMNOPQRSTUVWX
F$HIJKLMNOPQRSTUVWXYZABCDEFGHIJK                F$IJKLMNOP
F$JKLM                  F$KLMNOP
```

To control screen scrolling, use the -NO_WAIT option. To remind yourself of the syntax of the command, use the -HELP option.

Option	Meaning
[-NO_WAIT] [-NW]	Enables terminal screen scrolling. This option suppresses the —More— prompt that is otherwise given at the end of each 23 lines of display.

If you do not specify -NO_WAIT, PRIMOS prompts you before scrolling the terminal screen. To display the next screen of output, respond Y, YES, OK, NEXT, or press RETURN. To exit from the command, respond N, NO, Q or QUIT.

[-HELP] [-H] — Displays the command syntax. This HELP display is also printed if PRIMOS encounters an error while parsing the command.

Copying Over An In-Use EPF

No new options have been added to the COPY command for EPFs. However, you can use COPY to replace one EPF with another. The format of the command is:

```
COPY pathname [new-pathname] [options...]
```

pathname identifies the object you wish to copy (source object), and new-pathname identifies the destination and name of the copied object (target object). (The PRIMOS Commands Reference Guide describes the options to the COPY command.)

If the target object, identified by new-pathname, is open, the COPY command fails unless the target is an EPF. This example shows what happens when a user tries to copy the file ELEANOR to the open file FRANKLIN:

```
OK, COPY ELEANOR FRANKLIN
"FRANKLIN" already exists, do you wish to overwrite it? YES
File open on delete. Unable to delete file "FRANKLIN" (copy)
ER!
```

Using COPY to Replace an Open EPF File: At Rev. 19.4, the COPY command allows you to specify a target file that is open, provided that the target is an EPF. When you give the command, COPY performs a REPLACE operation in two stages:

1. First, COPY changes the name of an open EPF file, which you specify as the target object.
2. COPY then replaces this file with the file you specify as the source object.

For this REPLACE operation to work, the target object must be an EPF, and you must include its .RUN suffix on the command line.

To replace an open EPF file, you would give a command like:

```
COPY MYLIB>BETTER_EPF.RUN LIBRARIES*>OLD_EPF.RUN
```

where OLD_EPF is the (possibly open) file you want to replace and BETTER_EPF is the file you are putting in its place.

In this example, REPLACE works as follows:

1. The name of the target EPF file is changed. That is, the suffix .RUN is replaced by the suffix .RPn, where n is one of the 10 decimal digits in the range 0 to 9. In the above example, OLD_EPF.RUN might be renamed OLD_EPF.RP0.

2. The source EPF file is then copied to new-pathname. In the example, BETTER_EPF.RUN would be copied to LIBRARIES*>OLD_EPF.RUN

By default, COPY tells you that the target EPF file is open, and asks whether or not you want the REPLACE operation completed. COPY also displays the name of the replaced file. These messages are shown in the following example:

```
OK, COPY TIM>FORTRAN_IO_LIBRARY.RUN LIBRARIES*>==
Ok to replace EPF file LIBRARIES*>FORTRAN_IO_LIBRARY.RUN? YES
New version of EPF file FORTRAN_IO_LIBRARY.RUN now in place.
Old version of active EPF file now named FORTRAN_IO_LIBRARY.RP0.
```

To prevent the display of these messages, use the -NO_QUERY option of the COPY command.

Once you have replaced the EPF, anyone who invokes it will get the new version. However, if people were using the old version at the time that you replaced it, that version remains mapped into their address spaces. You may want to tell them what you have done, and suggest that they use REMOVE_EPF and invoke the new version if they wish.

REPLACE Files: PRIMOS does not delete REPLACE files when they are no longer mapped into the address space of any users. If you create REPLACE files, then you are responsible for deleting them once they are no longer needed.

Numbering: The suffix numbering sequence starts at .RP0, and continues to .RP9. This means there are up to 10 possible REPLACE files. If all possible files exist, PRIMOS asks you whether it can delete one of the REPLACE files which is not currently mapped to any user's address space, as shown in the following example.

```
OK, COPY TIM>LD.RUN CMDNC0>LD.RUN
Ok to replace EPF file CMDNC0>LD.RUN? YES
Ok to delete EPF file CMDNC0>LD.RP2? YES
New version of EPF file LD.RUN now in place.
Old version of active EPF file now named LD.RP2.
```

Using the -NO_QUERY option suppresses both the prompts shown in the above example.

If all ten REPLACE files are still mapped into the address spaces of some users, then the REPLACE operation cannot be completed, as the following example shows.

```
OK, COPY TIM>LD.RUN CMDNC0>LD.RUN
Ok to replace EPF file CMDNC0>LD.RUN? YES
EPF replace files are all in use.
Unable to complete file copy. (copy)
```

New User Attributes

The EPF product provides the user with many new features. Some of these features may indirectly cause the user to utilize more resources. This can happen completely without the users knowledge in some situtations. Therefore, this revision of the system allows the System Administrator to limit some of the command environment resources given to users on a per user basis.

This section is divided into three parts. The first part defines the command environment features that may be controlled. The second notes the EDIT_PROFILE interface to these features. The third lists the defaults.

Features

All of the command environment features that can be controlled on a per user basis are stored for each user in the System Administration Directory (SAD) as are user passwords and initial attach points. These new attributes are stored on a per project basis like the initial attach points rather than on a per system basis as are passwords.

The new attributes are divided into two types, segment usage limiters and command environment limiters. The segment limiters take the place of the NUSEG CONFIG parameter. The environment limiters are new.

Number of Dynamic Segments: The first new attribute is the number of dynamic segments that a user may receive in DTAR2. The dynamic segments are a set of segments into which PRIMOS command environment

information, all program EPFs, process and program class library EPFs, and the process, EPF, subsystem, and user class dynamic storage heaps are placed. The process, EPF, and subsystem heaps are used by PRIMOS while the user storage heap may be used by any programmer. These segments may be found at the high end of the user's DTAR2. The DTAR2 segments are defined within a range from '4000 to '5777.

Number of Static Segments: The second new attribute is the number of static segments a user may receive in DTAR2. These segments are used by static mode programs and may be found at the low end of the user's DTAR2. This attribute takes the place of the CONFIG parameter NUSEG.

Command Level Depth: The third new attribute is command level depth. This attribute controls the number of command levels that a user may receive. When logging in, a user is placed at command level one. Everytime a program takes an exception or a user enters a terminal quit, a new command level is entered. This attribute limits the number of times one of these events may occur by imposing a highest command level. At this command level, only certain designated PRIMOS commands may be used and terminal quits are ignored. This command level is the mini-command level.

Command Level Breadth: Command level breadth is used to control how many times programs can execute other programs. Remember that with EPFs, more than one program can be in memory at the same time.

Edit_Profile Interface

Edit_Profile has been modified to enable system and project administrators to set the new user attributes. This is done via the EDIT_PROFILE commands ADD_PROJECT, CHANGE_PROJECT, ADD_USER, and CHANGE_USER. The latter two must be used in conjunction with a project. This is the basically same as adding and changing initial attach points.

When a user logs in, the attributes for that user are set up. It is possible for a user to log in without having had these new attributes set up. If a user has had these attributes set up, then the user will be assigned these limits. If no limits can be found, then the default limits from the current project will be used. If the current project does not have these attributes set up, then the default system limits will be used.

Setting up these limits for a project is a two step process. First, EDIT_PROFILE will ask you to set up the project maximums. Then you will be asked to set up the project default. These defaults are given to users within this project that are not specifically assigned these attributes. This maximums are used as a check when setting the limits for either the project or any user within that project.

EDIT_PROFILE is now compatible in two ways with respect to these new attributes. Rev. 19.4 EDIT_PROFILE can use any SAD made from Rev. 19.2 to Rev. 19.4. The SAD does not have to contain this new profile information. Additionally, the data has been placed into the SAD in such a manner as to allow any revision of PRIMOS to read the SAD. Therefore, from Rev. 19.2 onward, compatability is supported both upwards and downwards with respect to revisions.

It is also possible to set these limits on a system wide basis via EDIT_PROFILE's CHANGE_SYSTEM_DEFAULTS command. The limits can be set to change the default limits that come with PRIMOS. Additionally, these system limits can be used to override the per user and per project limits by using EDIT_PROFILE's SYSTEM_DEFAULTS command.

Defaults

There are two kinds of defaults for these new attributes; project and system. If a user does not have these limits set on a per user basis, then the user will get the limits set for the current project. If the current project does not have limits set, then the user will get the limits set for the system. These limits come with PRIMOS and are compiled into it.

The default limits for projects are set in the SAD by either the System Administrator or by project administrators. The system limits from PRIMOS for Rev. 19.4 are:

Attribute	Recommend	Minimum	Maximum
Command levels	10	1	100
Live invocations per level	5	1	100
Private dynamic segments	40	16	504
Private static segments	40	8	496

Note that the total number of segments that a user may obtain is 512. This is 256 more segments than before. Also note that the number of dynamic segments takes precedence over the number of static segments.

Special notes about Dynamic Segments

The Linkage area of a program or library EPF is created using DTAR 2 segments in the '4000 range. The procedure code of an EPF is shared among users but each user will get his own copy of the linkage area. Once an EPF is mapped into the user's address space, it stays mapped in until it is explicitly removed via the REMOVE_EPF command or the command breadth is exceeded. At Rev. 19.4, there exist per-user attributes which limit the amount of virtual memory a user may acquire. One attribute limits the number of dynamic segments a user may own and

it can be set by a project/System Administrator via the EDIT_PROFILE subsystem. The default on this new attribute is set at 32 segments. If many EPFs have been mapped in to the user's address space (using per-user DTAR 2 segments), it is conceivable that the default limit of the number of dynamic segments is exceeded before another EPF can be mapped in. If this limit is exceeded the message "No more Segments" is displayed and the user can contact the System Administrator to increase his limit on dynamic segments. For more detailed information on these new attributes and dynamic virtual memory management, refer to PRIMOS documents for Rev. 19.4.

EDIT_PROFILE/USER 1 Dynamic and Static Segment Handling

It is now possible to change the default attributes for USER 1. To do this, the user must enter EDIT_PROFILE and issue the CHANGE_SYSTEM_DEFAULTS (CSD) command. In changing the system defaults, all users who do not have specific EPF attributes set will acquire the system default attributes. For users to acquire the new limits they must logout and login again. For USER 1 to acquire the new limits, the system must be cold started.

Calculation of Total Number of Static and Dynamic Segments

The sum of combined static and dynamic segments for normal users may not exceed 512. For USER 1, the sum of combined static and dynamic segments may not exceed 256. Therefore, if you issue the CSD command and change the number of static and dynamic segments to a number which exceeds 256, users will acquire limits equal to the value entered. For USER 1, the amount that you exceed the 256 limit by will be evenly subtracted from both the number of static segments and the number of dynamic segments. For example, if you enter 200 static segments and 58 dynamic segments, you will have exceeded the limit of 256 by 2. Therefore, the number of static segments will be changed to 199 and the number of dynamic segments will be changed to 57. If there is a wide range between the number of static segments entered and the number of dynamic segments entered, then the number of static and dynamic segments for USER 1 will be set to default values of 32 (given to the smaller segment value entered) and 224 (given to the larger segment value entered). For example, if you enter 33 static segments and 425 dynamic segments, the number of static segments will be changed to 32 and the number of dynamic segments will be changed to 224. The new system limits for USER 1 can be seen by issuing the LIST_LIMITS command at the supervisor terminal.

Programming Restrictions

When programming with EPFs, there are certain rules you must follow in order to meet the requirements of the EPF format. For example, since EPFs allow only reentrant code, your program cannot be self modifying. These rules have been put in place to ease the programmer's job. Sometimes this may mean changing a programming habit or two but, by using EPFs and following the rules, the end result will be a much better program.

This section is divided into four parts. The first part compares and contrasts EPFs versus static mode programs. The second part discusses the use of a few routines that were originally built for static mode programs. The third tells you what happens when you try running a remote EPF. The final part discusses some of the things you can and can not do via dynamic linking.

Compatability With Old Static Mode Programs: In general, most static mode programs can be made into EPFs by simply running them through BIND. The command environment for EPFs has been changed to be compatible with most of the program styles that can be found in static mode programs. This means that a program does not have to be fully structured to be an EPF. It is advised to program in a structured manner wherever possible. Remember, to the system your EPF looks like a subroutine. Therefore, if you write your programs such that they always return through their main entrypoint, you will have the most luck with EPFs.

All EPFs must be reentrant code. Reentrant code is always produced by Prime's compilers. This means that EPFs are generally better suited to high level languages rather than to PMA. It is possible to program in PMA and make EPFs as long as you do the following:

- Make sure your code is not self modifying.
- Make sure your non-constant data areas (linkage and common data) and your executable code are separated using the LINK pseudO-operation.
- If you have impure code, make sure to flag it as such via the PMA pseudo-operation, SEG IMPURE.

One other area to watch out for when using EPFs is the interaction of command lines and EPFs. If you have an EPF which is to be passed by the command line as an argument, you cannot use the RDTK$$ interface. You should use the passed command line argument together with the CL$PIX routine. If your EPF does not take the command line argument, then is is all right to use the RDTK$$ interface. The system detects whether or not to pass the command line to your program. This is another example of how the EPF product has been made compatible with static mode programs.

Using The Static Mode Interfaces: A few routines that have been in the system for a long time were specifically built to allow termination of static mode programs. They are:

- EXIT
- ERRPR$ with the K$NRIN key
- ERRSET

If you call these routines from an EPF, the system will recognize this fact and handle the termination in a manner different from handling termination of a static mode program. In other words, no adverse effects will be felt by an EPF that uses these routines. It must be noted that the system signals STOP$ for EPFs when these routines are called. It then cleans up the EPF's stack. The use of these routines from inside of EPFs is strongly discouraged for three reasons:

- The signalling of the STOP$ condition happens unknown to the EPF. It is possible that the program may have set up a handler for this condition. This handler may then get invoked at the wrong time.
- The signalling of the STOP$ condition prevents program restart.
- Use of these routines promotes unstructured code.

Since FTN does not allow a return from the MAIN module, the last statement within the MAIN module should be a STOP statement. The same is true for COBOL and CBL. All other languages should use a return statement. In addition, the EPF may itself use the SIGNL$ routine to signal the STOP$ condition.

Use of EXIT to pause a program is also disallowed. To pause a program use a PAUSE statement, use the SIGNL$ routine with the PAUSE$ condition, or use ERRPR$ with the K$SRIN key. Use of ERRPR$ with the K$IRIN key is always allowed.

Remote EPFs: Running a program EPF from a remote partition is allowed although the system will not use VMFA to read the EPF into memory. Rather, the EPF will be copied onto the paging partition and will be read from there. The startup time for a remote EPF is longer than that for a local EPF because of the copying needed. Also, since the EPF will be on the paging partition, it will not run as fast as an EPF using VMFA. This is because VMFA is not always in contention for the same disk drive which contains the paging partition.

A remote program EPF once copied onto the paging partition can do anything any other EPF can do such as call an external program, gets its arguments directly from the system, and return directly to the system.

Dynamic Linking: Once again there are certain rules to be followed when building and using libraries. The following table shows which links are valid:

FROM/TO	PROGRAM	PROGRAM CLASS LIBRARY	PROCESS CLASS LIBRARY	STATIC MODE LIBRARY
PROGRAM	NA	VALID	VALID	VALID
PROG. LIB.	NA	VALID	VALID	VALID
PROC. LIB.	NA	INVALID	VALID	INVALID
STAT. LIB.	NA	VALID	VALID	VALID

In addition, there is one other rule:

> Whenever a static mode library is in use, linking to it is only valid within the same program or library invocation.

This means that only one active invocation of a static mode library is allowed at any given time. In other words, if a program uses the DBMS static mode library and then calls another program which wants to use DBMS, the second program will not be permitted to use DBMS because it is active only for the first program. So watch out for the libraries being used by external programs that haved been called by other programs.

It should also be mentioned that the changing of levels due to a terminal quit or an exception does not free up an in-use static mode library. The same restrictons apply here as for when programs call programs.

Error Detection And Handling

This section describes any policies used for reporting or handling error conditions. Error processing may be broken down into three areas:

- Returning PRIMOS defined error codes to application software
- Displaying error text to a user's terminal and either continuing processing or re-initializing the user's command environment
- Signalling a condition to be handled by user software or, in most cases, by the command environment (system defined condition)

Exceeding Resource Limits

Users of the EPF product will run up against both system and per-user resource limitations. System limits will be set up by cold start CONFIG directives and dedicated System Administrator commands. Per-user limits will be set up via project administrator or System Administrator control of the user profile mechanism.

System Resources: The primary system resource for the product is virtual memory segments. Two categories of segments exist: non-VMFA and VMFA segments. Additionally, paging disk space may be a scarce resource.

- If there are no non-VMFA segments available, users may not perform useful work until a system segment is freed by some user.

- If the system runs out of those segments allocated to EPF procedure code (VMFA segments), users who wish to execute EPFs may have to wait until another VMFA segment is freed. Users are cautioned that each installed EPF library will use up at least one VMFA segment.

- If the system runs out of paging disk space, the user may have to wait until more disk space is made available as user's de-activate segments or the user may have to contact the System Administrator and request more paging space be allocated for the next system cold start.

In each case, users of the EPF product will be returned an error code indicating the cause of the problem. Under most conditions, the program will need to prompt the user for some help in freeing the scarce resource.

User Resources: There exist per-user resource limits on the command environment. The primary goal is to limit the effect any one user may have on the overall performance of the system. Limits exist on both the depth and breadth of the command environment and the number of both static and dynamic private segments.

- Upon reaching the "high water mark" for the command environment, a user will be allowed to issue only a restricted set of commands. This "high water mark" is what has been previously referred to as the mini-command level. This set of commands will be displayed once a user exceeds the command level high water mark. The command LIST_MINI_COMMANDS, (LMC), will also display these commands. Please refer to earlier sections of this document for more information on the mini-command level.

- If the per-user limit on the number of simultaneous program invocations at one command level; that is, the breadth of the command environment, is exceeded, the error code "E$ECEB" is

returned to the user software. It is the responsibility of the application programmer to perform any necessary cleanup operations, such as closing of file units.

In order to minimize this difficulty, there exists a program callable interface in order to determine this command environment limit.

```
dcl rd$ced entry (fixed bin(15));

call rd$ced(current_command_environment_breadth);
```

- If a user has run out of private, dynamic segments, the user will be notified of the resource problem in one of three possible methods:

    1. The error code E$NMTS may be returned to the running program.

    2. The condition name "STORAGE" may be signalled.

    3. The user's Ring 3 environment may be re-initialized.

  Again, in most cases, the problem will need to be solved by the user at command level.

- If a user runs out of private, static segments, the condition name ILLEGAL SEGNO$ will be signalled within the user process.

If a user runs out of resources (segments), the user will be able to display the memory resources used by program invocations in the address space with either the LIST_EPF or LIST_SEGMENT commands.

For most cases, user segments are also using up system-wide resources. Therefore, freeing up per-user segments will help alleviate any system-wide resource scarcity. There no longer exists any functionality to de-allocate specific dynamic segments such as with the DELSEG command. The REMOVE_EPF command will allow a user to de-allocate any memory resources used by an active EPF. The RELEASE_LEVEL command will allow users to return to previously suspended command levels, thereby releasing any memory resources utilized at the command levels being released. The ICE command will allow a user to completely re-initialize their command environment, causing all of a users's private, dynamic segments to be de-allocated.

Linkage Fault Handling

The information provided at command level to a user in the advent of a linkage fault includes the name of the faulted entrypoint and the name of the procedure from which the fault had originated (if the name is attainable).

Attempting Invalid Link Types: The dynamic linking mechanism does not correct attempts to create unsupported link types but detects such cross references among library classes and library types. Unsupported dynamic link types are detected by the dynamic linking mechanism and an appropriate system defined condition is signalled. User software may set up on-units to handle the specified conditions. About all that may be done is to display an informative error message and temporarily abort the program session. The only permanent solution is:

- If the invalid link type is related to EPF library class, convert the EPF library to the proper class.
- If the invalid link type is related to an in use static-mode library, convert the static-mode library to an EPF library.

Dynamic links to static-mode libraries that are currently in use by programs at lower command levels are allowed. Programs at previous command levels that are currently using these libraries will be marked as not restartable. This is similar to invoking a static image over a suspended static image. An informative error message will be displayed if an attempt is made to restart a program at a previous command level that is using these libraries.

New Error Codes: The following new error codes have been added for the EPF product:

e$epft:	Invalid EPF type
e$epfs:	Invalid EPF search type
e$iltd:	Invalid EPF LTD linkage descriptor
e$ilte:	Invalid EPF LTE linkage descriptor
e$eceb:	Exceeding command environment breadth
e$epfl:	EPF file exceeds file size limit
e$nta:	EPF file not active for user
e$swps:	EPF file suspended within current program session
e$swpr:	EPF file suspended within this process
e$adcm:	System Administrator command only

New Keys: The following new keys have been added for the EPF product:

k$copy:	Copy EPF file into temporary segments
k$dbg:	Map DBG information into memory from EPF
k$initall:	Initialize all of the linkage areas
k$reinit:	Only reinitialize linkage areas
k$invk:	Invoke and do not de-allocate EPF from memory
k$invk_del:	Invoke and delete EPF
k$restore_only:	Restore but do not invoke EPF
k$frc_del:	Force terminate EPF
k$no_frc_del:	Do not force terminate EPF
k$vmr:	Open EPF file for reading
k$any:	Any segment(s) acceptable

k$dupl:	Duplicate segments requested
k$cnsc:	Consecutive segments required
k$r:	Read access only
k$rx:	Read/execute access only

New System Conditions: The following new system defined condition name has been added for the EPF product:

1) LINKAGE_ERROR$:
   An error occurred while attempting to resolve a dynamic link. The type of error will be obvious from the text of the error message being displayed.

New System Error Messages: The following EPF error messages have been added to the file EPF_ERROR_TABLE in the UFD SYSOVL:

- dealloc_procedure_segs

  An error was encountered while attempting to de-allocate EPF procedure segments for

- circular_epf_level_cache

  Internal EPF error: the EPF level cache has become circularized. Please contact your System Administrator.

- circular_list_of_epfs

  Internal EPF error: the list of active EPFs for this process has become circurlarized. Please contact your System Administrator.

- circular_entry_search_list

  Internal EPF error: the ENTRY search list for this process has become circularized. Please contact your System Administrator.

- epf_smt_not_found

  Internal EPF error: the segment mapping table for EPF file %v could not be retrieved. Please contact your System Administrator.

- epf_storage_class_corrupted

  Internal dynamic storage error: the EPF dynamic storage class has been corrupted. Please contact your System Administrator.

- user_storage_class_corrupted

  Internal dynamic storage error: the USER dynamic storage class has been corrupted. Please contact your System Administrator.

- level_storage_class_corrupted

  Internal dynamic storage error: the LEVEL dynamic storage class has been corrupted. Please contact your System Administrator.

- program_session_depth_invalid

  Internal EPF error: the depth of the program session for this user is invalid. Please contact your System Administrator.

- command_level_depth_exceeded

  You have exceeded your maximum number of command levels.

  You are now at mini-command level. Only the commands shown below are available. Of these, RLS -ALL should return you to command level 1. If it does not, type ICE. If this problem recurs, contact your System Administrator.

  Valid mini-commands are:

- circular_command_level_list

  Internal EPF error: the command environment level structure is invalid. Please contact your System Administrator.

- circular_epf_smt_active_ents

  Internal EPF error: the EPF segment mapping table for %v has become invalid. Please contact your System Administrator.

System Administrator Information

For the System Administrator there are four main areas of concern, CONFIG parameters, the Edit_Profile interface, the system entrypoint search list, and the LIBRARIES* directory.

CONFIG Parameters: Two CONFIG parameters are affected by the EPF product, NUSEG and NVFMS.

NUSEG, the number of segments that a user may acquire, no longer applies. If it is used a diagnostic will be printed. It has been superceded by the new number of dynamic segments and number of static segments user attributes.

NVMFS, the number of read only VMFA segments, is new with the EPF product. EPFs are placed into read-only VMFA segments. This parameter controls the total number of these segments that may be active on a system. When setting this number remember that these segments are shared between users. The default for NVFMS is 64 ('100). The minimum is 0 and the maximum is 1024 ('2000).

EDIT_PROFILE Interface: The new user attributes have been discussed previously. System Administrators and project administrators should understand that none of these new features need to be used. PRIMOS will work perfectly well using the predefined system defaults. If you do want to make use of these features, EDIT_PROFILE must be used to set up these new user resource limits. The EDIT_PROFILE commands that are importrant are:

```
ADD_PROJECT
CHANGE_PROJECT
ADD_USER
CHANGE_USER
CHANGE_SYSTEM_DEFAULTS
SYSTEM_DEFAULTS
```

The EDIT_PROFILE part of the EPF product has been built to insure both upward and downward compatability. This means that any Rev. 19.2 or later SAD can be used on any Rev. 19.2 or later system no matter what attributes are used. This capability will be maintained in the future.

Entrypoint Search List: The entrypoint search list is used to tell PRIMOS where to resolve dynamic links. A default list will come with PRIMOS in the SYSTEM directory. It contains rules that tell the system when to search the static mode libraries and when to search the system EPF libraries. It also tells where these libraries can be found. It is an editable ASCII file.

If you need to change the order of this list or need to add new items to it, you can simply edit the file. Remember, currently logged in users will not reflect the changes you make until either they login again, initialize their command environment, or issue a SET_SEARCH_RULES command.

LIBRARIES*: LIBRARIES* is the directory which contains the EPF libraries provided by Prime. It is new with the EPF product. It is also a good place to put any public EPF libraries that you create.

Converting Old Static Mode Programs To EPFs

As we have stated all along, the EPF product is fully compatible with the old static mode programs. In fact, it is so compatible that most static mode programs can be converted into EPFs by simply using BIND instead of SEG. Of course, there are a few exceptions and some catches that can make conversion a little bit harder.

This section will present you with the reasons why you should convert your present programs into EPFs. The kind of programs that cannot be converted will be discussed and you will be provided you with a step-by-step example of how to convert a static mode program into an EPF.

Why Would You Want To Convert?: The major reason to convert your programs to EPFs is the ease with which EPFs are built and maintained. Not having to use SEG is a real blessing. BIND is a much cleaner interface.

The second most important reason to convert your programs to EPFs is that, by converting, you make them callable from other programs. Remember, EPFs will not overlay one another. They are maintained in memory by the operating system.

Another reason to convert your programs to EPFs is for expandability. The nature of the interaction of an EPF program image with the system allows you to forget about how big a program is, where its data goes, and so forth. You can write as much code as you want without having to worry about space limitations. You don't have to worry about whether your code will fit into the memory allocated to it or whether another segment must be added by the programmer.

Furthermore, if you do a good job converting your programs to EPFs, the result will be well-structured code which is much easier to maintain than unstructured code. Any savings in the maintainence phase of a program may well be worth the effort of conversion.

Finally, you may wish to convert your program into an EPF as the first step in building up an EPF library of similar programs.

What You Can Not Convert: The main class of static mode programs which are not amenable to conversion are shared static mode programs which use the PBECB option. PBECB shares link frames with executable code. This mixes writable storage with read-only storage. EPFs can only have read-only storage. Therefore, you can not convert these programs unless you can drop the -PBECB option.

The second class of programs which can not be converted are programs in which some linkage has been made public. By public is meant that it resides in a shared segment. This is usually done to minimize the working set of a program. The EPF product presently does not have the ability to put linkage into shared segments.

The next class of programs which can not be converted easily are typical PMA programs. Most often you will find executable code and data well mixed within a PMA program. This makes it impossible for BIND to separate code and data into pieces that it can make into an EPF. It should be added that some PMA programs may be converted with some effort. It all depends on how well the programs were originally written.

Another kind of program that can not be converted is the one which has self-modifiying code. This means that code will change while the program is running. EPFs must be pure code. Therefore, these programs cannot be converted.

Finally, some programs have been built to fit into a very small amount of memory for one reason or another. If they were converted into EPFs, they might take up too much space. They will definitely find themselves in more than one segment with the program in one segment and linkage in another.

Sample Conversion: This section will take you through the conversion of R-mode and S-mode programs into an EPF. The conversion will take place in a step-by-step manner. The approximate time it took to perform the conversion will also be noted. The example will convince you that it is worth your time to convert some programs into EPFs.

The Static Mode Program: The following is a static mode program which determines if a file exists in the current directory or, optionally, in a UFD. It is a command with this syntax:

```
FILE_EXISTS <filename> [<ufdname>]
```

The program verifies that the proper arguments have been entered, tells the user when any errors were made, and states whether or not the file exists.

```
file_exists: proc;

dcl info (8) fixed bin,          /* info vector for rdtk$$ */
    filename char(32),           /* file to check existance of */
    filename_len fixed bin,
    ufdname char(32),            /* where to look */
    ufdname_len fixed bin,
    ercode fixed bin,            /* standard error code */
    pathname char(65),           /* pathname of file to look for */
    pathname_len fixed bin,
    chrpos (2) fixed bin,        /* info vector for tsrc$$ */
    unit fixed bin,              /* unit file opened on */
    type fixed bin;              /* type of file */
```

```
%replace rdtk_key by 1,        /* read next token and convert to
                                  uppercase */
        k$nrtn by 0,           /* no return */
        e$misa by 170,         /* missing argument */
        k$exst by 6;           /* check file existance */

dcl rdtk$$ entry(fixed bin, (8) fixed bin, char(*), fixed bin,
                 fixed bin),
    errpr$ entry(fixed bin, fixed bin, char(*), fixed bin, char(*),
                 fixed bin),
    tsrc$$ entry(fixed bin, char(*), fixed bin, (2) fixed bin,
                 fixed bin, fixed bin),
    tnoua entry(char(*), fixed bin),
    tnou entry(char(*), fixed bin),
    exit entry;

/* ***************************************************************/

/* Get the name of the file and its length. */

call rdtk$$(rdtk_key, info, filename, 32, ercode);
filename_len = info(2);

/* Make sure file argument was given and is ok. */

if ercode = 0 & filename_len = 0 then
  ercode = e$misa;
if ercode = 0 then
  do;                          /* file arg is ok */

/* Get the name of the ufd and its length. */

    call rdtk$$(rdtk_key, info, ufdname, 32, ercode);
    ufdname_len = info(2);

/* Make sure ufd argument is ok if given. */

    if ercode = 0 then
      do;                      /* ufd arg is ok */

/* Make a pathname up out of the ufd name and the file name.
   Remember, the ufd name may not have been given. */

        if ufdname_len > 0 then
          do;
            pathname = substr(ufdname, 1, ufdname_len) || '>' ||
                       substr(filename, 1, filename_len);
            pathname_len = ufdname_len + 1 + filename_len;
          end;
        else
          do;
            pathname = substr(filename, 1, filename_len);
            pathname_len = filename_len;
          end;
```

```
/* Now set up for and see if file exists. */

        chrpos(1) = 0;
        chrpos(2) = pathname_len;
        call tsrc$$(k$exst, pathname, unit, chrpos, type,
                    ercode);

/* Tell user results. */

        if ercode = 0 then
          do;
            call tnoua(filename, filename_len);
            call tnou(' exists.', 8);
          end;
        else
          call errpr$(k$nrtn, ercode, filename, filename_len,
                      'FILE_EXISTS', 11);

      end;                          /* ufd arg is ok */
    else
      call errpr$(k$nrtn, ercode, 'ufd argument', 13,
                  'FILE_EXISTS', 11);

  end;                              /* file arg is ok */
else
  call errpr$(k$nrtn, ercode, 'file argument', 13,
              'FILE_EXISTS', 11);

call exit;

end;
```

The First Conversion Step: The first conversion step is nothing more than using BIND on the program's object instead of using SEG. The compatability built into the EPF product allows this program to execute as an EPF. To BIND this program you would enter this command line:

```
BIND FILE_EXISTS -lo FILE_EXISTS.BIN
```

BIND produces FILE_EXISTS.RUN which you could then RESUME. The time needed here to convert is equal to the time needed to run BIND.

The Second Conversion Step: The second conversion step involves remembering that EPFs are called as subroutines of the operating system and should directly return to the system. The final call to EXIT in the program prevents this direct return so the next conversion step involves removing the call to EXIT. This is demonstrated below. Note that the declaration for EXIT has also been removed.

```
file_exists: proc;

dcl info (8) fixed bin,          /* info vector for rdtk$$ */
    filename char(32),           /* file to check existance of */
    filename_len fixed bin,
    ufdname char(32),            /* where to look */
    ufdname_len fixed bin,
    ercode fixed bin,            /* standard error code */
    pathname char(65),           /* pathname of file to look for */
    pathname_len fixed bin,
    chrpos (2) fixed bin,        /* info vector for tsrc$$ */
    unit fixed bin,              /* unit file opened on */
    type fixed bin;              /* type of file */

%replace rdtk_key by 1,          /* read next token and convert to
                                    uppercase */
        k$nrtn by 0,             /* no return */
        e$misa by 170,           /* missing argument */
        k$exst by 6;             /* check file existance */

dcl rdtk$$ entry(fixed bin, (8) fixed bin, char(*), fixed bin,
                 fixed bin),
    errpr$ entry(fixed bin, fixed bin, char(*), fixed bin, char(*),
                 fixed bin),
    tsrc$$ entry(fixed bin, char(*), fixed bin, (2) fixed bin,
                 fixed bin, fixed bin),
    tnoua entry(char(*), fixed bin),
    tnou entry(char(*), fixed bin);

/* ************************************************************/

/* Get the name of the file and its length. */

call rdtk$$(rdtk_key, info, filename, 32, ercode);
filename_len = info(2);

/* Make sure file argument was given and is ok. */

if ercode = 0 & filename_len = 0 then
  ercode = e$misa;
if ercode = 0 then
  do;                            /* file arg is ok */

/* Get the name of the ufd and its length. */

    call rdtk$$(rdtk_key, info, ufdname, 32, ercode);
    ufdname_len = info(2);

/* Make sure ufd argument is ok if given. */

    if ercode = 0 then
      do;                        /* ufd arg is ok */
```

```
/* Make a pathname up out of the ufd name and the file name.
   Remember, the ufd name may not have been given. */

        if ufdname_len > 0 then
          do;
            pathname = substr(ufdname, 1, ufdname_len) || '>' ||
                       substr(filename, 1, filename_len);
            pathname_len = ufdname_len + 1 + filename_len;
          end;
        else
          do;
            pathname = substr(filename, 1, filename_len);
            pathname_len = filename_len;
          end;

/* Now set up for and see if file exists. */

        chrpos(1) = 0;
        chrpos(2) = pathname_len;
        call tsrc$$(k$exst, pathname, unit, chrpos, type,
                    ercode);

/* Tell user results. */

        if ercode = 0 then
          do;
            call tnoua(filename, filename_len);
            call tnou(' exists.', 8);
          end;
        else
          call errpr$(k$nrtn, ercode, filename, filename_len,
                      'FILE_EXISTS', 11);

      end;                          /* ufd arg is ok */
    else
      call errpr$(k$nrtn, ercode, 'ufd argument', 13,
                  'FILE_EXISTS', 11);

  end;                              /* file arg is ok */
else
  call errpr$(k$nrtn, ercode, 'file argument', 13,
              'FILE_EXISTS', 11);

end;
```

This conversion took only about five minutes. To make an EPF out of this source code, you need to compile the source code and execute BIND which takes another few minutes.

Conversion Step Three: If you take a careful look at what we now have, you will encounter calls to the error printing routine, ERRPR$. These calls use the K$NRTN key. This key causes ERRPR$ to never return to the program. If this is the case, the program cannot directly return to the system. Since it is better for EPFs to directly return, change the key to K$IRTN.

```
file_exists: proc;

dcl info (8) fixed bin,          /* info vector for rdtk$$ */
    filename char(32),           /* file to check existance of */
    filename_len fixed bin,
    ufdname char(32),            /* where to look */
    ufdname_len fixed bin,
    ercode fixed bin,            /* standard error code */
    pathname char(65),           /* pathname of file to look for */
    pathname_len fixed bin,
    chrpos (2) fixed bin,        /* info vector for tsrc$$ */
    unit fixed bin,              /* unit file opened on */
    type fixed bin;              /* type of file */

%replace rdtk_key by 1,          /* read next token and convert to
                                    uppercase */
        k$irtn by 0,             /* immeadiate return */
        e$misa by 170,           /* missing argument */
        k$exst by 6;             /* check file existance */

dcl rdtk$$ entry(fixed bin, (8) fixed bin, char(*), fixed bin,
                 fixed bin),
    errpr$ entry(fixed bin, fixed bin, char(*), fixed bin, char(*),
                 fixed bin),
    tsrc$$ entry(fixed bin, char(*), fixed bin, (2) fixed bin,
                 fixed bin, fixed bin),
    tnoua entry(char(*), fixed bin),
    tnou entry(char(*), fixed bin);

/* ******************************************************/

/* Get the name of the file and its length. */

call rdtk$$(rdtk_key, info, filename, 32, ercode);
filename_len = info(2);

/* Make sure file argument was given and is ok. */

if ercode = 0 & filename_len = 0 then
  ercode = e$misa;
if ercode = 0 then
  do;                            /* file arg is ok */
```

```
/* Get the name of the ufd and its length. */

    call rdtk$$(rdtk_key, info, ufdname, 32, ercode);
    ufdname_len = info(2);

/* Make sure ufd argument is ok if given. */

    if ercode = 0 then
      do;                          /* ufd arg is ok */

/* Make a pathname up out of the ufd name and the file name.
   Remember, the ufd name may not have been given. */

        if ufdname_len > 0 then
          do;
            pathname = substr(ufdname, 1, ufdname_len) || '>' ||
                       substr(filename, 1, filename_len);
            pathname_len = ufdname_len + 1 + filename_len;
          end;
        else
          do;
            pathname = substr(filename, 1, filename_len);
            pathname_len = filename_len;
          end;

/* Now set up for and see if file exists. */

        chrpos(1) = 0;
        chrpos(2) = pathname_len;
        call tsrc$$(k$exst, pathname, unit, chrpos, type,
                    ercode);

/* Tell user results. */

        if ercode = 0 then
          do;
            call tnoua(filename, filename_len);
            call tnou(' exists.', 8);
          end;
        else
          call errpr$(k$irtn, ercode, filename, filename_len,
                      'FILE_EXISTS', 11);

      end;                         /* ufd arg is ok */
    else
      call errpr$(k$irtn, ercode, 'ufd argument', 13,
                  'FILE_EXISTS', 11);

  end;                             /* file arg is ok */
else
  call errpr$(k$irtn, ercode, 'file argument', 13,
              'FILE_EXISTS', 11);

end;
```

This conversion took another five minutes. Again, you must now compile and BIND the program.

Please note that many programs do not have the innate structure of FILE_EXISTS. Without the structure, this step may take much longer. Remember, this step is not needed; the program will run without it. It all depends on how much work you wish to put into your conversions. Converting and structuring your programs will make future maintainence much easier.

Conversion Step Four: In this step it is noted that EPFs, as subroutines of the system, are passed arguments. One of the arguments is always the command line. Therefore, the program is modified to accept this argument. In doing so, the program is changed to call the command line parsing routine CL$PIX. This conversion took about an hour.

```
file_exists: proc(command_line, command_status, command_name);

dcl command_line char(80) var,
    command_status fixed bin,
    command_name char(11) var;

dcl pix char(18) var static init('entry; entry; end;'), 1 struc,
     2 filename char(32) var, /* file to check existance of */
     2 ufdname char(32) var,  /* where to look */
    pix_idx fixed bin,
    bad_idx fixed bin,
    pathname char(65) var,    /* pathname of file to look for */
    chrpos (2) fixed bin,     /* info vector for tsrc$$ */
    unit fixed bin,           /* unit file opened on */
    type fixed bin;           /* type of file */

dcl 1 cvs based,              /* overlay for a varying string */
     2 len fixed bin,
     2 chars char(1);

%replace cl$pix_key by '0002'b4,
         k$irtn by 0,         /* immeadiate return */
         e$misa by 170,       /* missing argument */
         k$exst by 6;         /* check file existance */

dcl cl$pix entry(bit(16) aligned, char(*) var, ptr, fixed bin,
                 char(*) var, ptr, fixed bin, fixed bin, fixed bin),
    errpr$ entry(fixed bin, fixed bin, char(*), fixed bin, char(*),
                 fixed bin),
    tsrc$$ entry(fixed bin, char(*), fixed bin, (2) fixed bin,
                 fixed bin, fixed bin),
    tnoua entry(char(*), fixed bin),
    tnou entry(char(*), fixed bin);

/* ************************************************************/
```

```
/* Get the name of the file and the ufd. */

call cl$pix(cl$pix_key, command_name, addr(pix), length(pix),
            command_line, addr(struc), pix_idx, struc_idx,
            command_status);

/* Make sure arguments are ok and file argument was given. */

if command_status = 0 & length(struc.filename) = 0 then
  command_status = e$misa;
if command_status = 0 then
  do;                              /* file arg is ok */

/* Make a pathname up out of the ufd name and the file name.
   Remember, the ufd name may not have been given. */

    if length(struc.ufdname) > 0 then
      pathname = struc.ufdname || '>' || struc.filename;
    else
      pathname = struc.filename;

/* Now set up for and see if file exists. */

    chrpos(1) = 0;
    chrpos(2) = length(pathname);
    call tsrc$$(k$exst, addr(pathname) -> cvs.chars, unit,
                chrpos, type, command_status);

/* Tell user results. */

    if command_status = 0 then
      do;
        call tnoua(addr(struc.filename) -> cvs.chars,
                   length(filename));
        call tnou(' exists.', 8);
      end;
    else
      call errpr$(k$irtn, command_status,
                  addr(struc.filename) -> cvs.chars,
                  length(filename), 'FILE_EXISTS', 11);
  end;                             /* file arg is ok */
else
  call errpr$(k$irtn, command_status, 'file argument', 13,
              'FILE_EXISTS', 11);
end;
```

Note that by using CL$PIX, the argument-checking logic within the program was simplified. Also, since CL$PIX uses varying length strings by default, our program was converted to use the same type and the extra SUBSTR calls needed to remove the blank padding from the end of the fixed length strings was eliminated.

Remember, you must still compile and BIND the program.

CHAPTER 3

PRIMOS and Utilities

PRIMOS AND PRIMOS II

PRIMOS

RECOVERABLE MACHINE CHECK FUNCTIONALITY

Current Prime processors allow correctable main-memory parity errors to be retried without affecting the operation of the system. However, parity errors in other places in the processor, such as the main-memory cache, cause the system to immediately halt. Some of the new processors now under development allow other transient errors to be corrected thereby providing improved system availability.

With most existing Prime processors, PRIMOS halts the system if there is a parity error in either the cache or the STLB. This is because previous Prime processors may have unrecoverably damaged the "machine state" as a consequence of taking the check or attempting to use the bad data. This can be true even if one attempts to warmstart the system. Therefore, the PRIMOS action is to attempt to log the error and then halt the system.

With the introduction of new hardware, logic will be able to effectively correct the problem without affecting the processor microcode. When the hardware detects a parity error in either the cache or the STLB, it forces a cache or STLB "miss" which causes the

erroneous location to be reloaded from main memory. This operation is completely transparent to the microcode.

The obvious reason for implementing recoverable machine checks is for higher availability on future processors. It is felt that a very high percentage of cache parity errors are transient in nature. Thus, allowing complete recovery from these errors can markedly increase the amount of time the system is operational.

Future machines will have faster processors than the P9950 and the main-memory cache, the branch cache, and the STLB may be larger. The increase in size of these elements could be accomplished by using denser logic. There is some concern that these components may experience a higher rate of failure than current parts and thus a way to improve the situation is desired. The concept of recoverable machine checks is the answer. It should be emphasized that cache and STLB parity errors are not especially frequent today.

The functionality of the addition of correctable machine checks is completely transparent to a typical user. The system will not stop when a correctable parity error occurs. In fact, except for the log entry, there will be no external indication that an error occurred at all.

However, a system administrator or field service person would be able to look at the PRIMOS event log files and tell that the system had been having correctable machine checks. The same information that is currently returned on unrecoverable machine checks will still be returned to one examining the machine history using either LOGPRT or PRINT_SYSLOG. The only difference is that a corrected machine check will say so.

Another point about recoverable machine check handling should be made. It can be relatively expensive to log large numbers of these errors. If all such errors were logged, it is conceivable that some significant performance degradation would be experienced. In fact, the logging of errors would degrade performance far more than simply correcting the original problem. This problem will be addressed similar to correctable memory parity errors. A count of the number of correctable machine checks will be kept in PRIMOS and when this number exceeds a certain arbitrary number, 1024, PRIMOS will cause the hardware to suppress reporting of recoverable machine checks.

In order to support the recoverable machine check interface, changes had to be made to the machine check handler in PRIMOS and to those routines that output the event log information generated by PRIMOS.

Since a conscious decision was made to choose the approach that had the least effect on the check handling code, the changes here are quite simple. A new check header has been put into segment 4 at location 320. This check header invokes code that very easily merges with the code that already counts correctable main memory parity errors and then restarts PRIMOS. The only change was to add a separate counter for corrected machine checks.

Modifications to LOGPRT and PRINT_SYSLOG to support ECCU

There are two programs that print out the contents of the PRIMOS event log files. These programs have the same functionality. Both LOGPRT and PRINT_SYSLOG have been changed to recognize recoverable machine checks. They have been modified so that the routine that decodes the contents of DSWSTAT checks to see if bit 26 of DSWSTATL is set. If it is, these utilites have been modified to indicate that the machine check was recovered.

An error will be put into the NETLOG and PRINT_SYSLOG when a bad HDX restart packet is received by the DTE.

CONFIGURATION DIRECTIVES

LOGBAD

If a login attempt is unsuccessful (due to an unrecognized userid or a bad password or project) and the directive:

```
LOGBAD YES
```

appears in the CONFIG file, one of the following messages is printed at the supervisor terminal, depending on the login type.

Login Was Local and Terminal Line Number is Known:

```
Failed login <day-of-week> <date> <time>: line# <line>,
UserId <userid>.
```

Login Was Local and Terminal Line Number is Unknown:

```
Failed login <day-of-week> <date> <time>: line# UNKNOWN,
UserId <userid>.
```

Note

This case would only occur if the userid did not appear in LWORD, a table mapping userid's to terminal line numbers. This case should never occur.

Login Was Remote and Remote Node Was a PRIMENET Node:

Failed login <day-of-week> <date> <time>: UserId <userid>,
from PRIMENET system <primenet_node_name>.

Login Was Remote and Remote Node Was Not a PRIMENET Node:

Failed login <day-of-week> <date> <time>: UserId <userid>,
from X.25 address <X.25_address>.

Login Was Remote and Remote Login Was From an Unknown Node:

Failed login <day-of-week> <date> <time>: UserId <userid>,
from UNKNOWN network address.

Notes

- This case should never occur.
- If the failed login attempt occurred before the date and time was set, then "<day-of-week> <date> <time>" is replaced by "At system startup".

The appearance of LOGBAD NO in CONFIG results in no failed login messages appearing on the supervisor terminal. The absence of LOGBAD YES|NO in CONFIG is equivalent to the presence of LOGBAD NO.

NEW AMLC PROTOCOL

TT8BIT

TT8BIT, the eight bit protocol, allows for recognition of character sets which require the use of 8 bits to represent each character. It was written to provide support for the Arabic character set. Essentially, the protocol behaves in the same manner as the TTY protocol with the exception that the eighth bit (ASCII parity bit) is not forced on for each character input from the terminal. This is to allow the eighth bit to be used as part of the character. All control characters are handled in the same manner as the TTY protocol.

The protocol requires special firmware support from the terminal and may be run only from locally attached terminals. Remote login can not be used. The special terminal sends 8 bits of character and 1 bit of parity. This requires that parity be enabled on the Prime as specified below.

In order to use this protocol the user must specify the protocol name TT8BIT in the AMLC command and set bit 13 of the line configuration word to 0 to enable parity. For example:

```
AMLC TT8BIT 0 2403 /* Line 0, odd parity enabled, 9600 baud.
```

REVERSE FLOW CONTROL ON THE ICS1 AND ICS2 CONTROLLERS

PRIMOS support is provided for reverse flow control on the ICS1 and ICS2 controllers. Users can now enable this feature via the AMLC command for use while running certain block mode terminals on the ICS1 and ICS2.

ADDITIONAL ASYNCHRONOUS LINE SUPPORT

At Rev. 19.4, the maximum number of asynchronous lines which can be supported on a single system has been raised from 128 to 256. On a single system there may be a mix of AMLC, ICS1, and ICS2 asynchronous lines. There has not been a change in the number of AMLC controllers that may be in a single system; the limit remains at eight AMLC controllers and, therefore, 128 AMLC lines. PRIMOS now supports up to four ICS controllers, which may be any mix of ICS1 and ICS2.

Since there are only 255 processes supported by PRIMOS, one of which is reserved for the supervisor terminal, only 254 asynchronous lines may be configured for user login. The remaining two lines can be configured as assignable lines.

There have been no syntax changes in the commands or directives applicable to asynchronous lines as a result of this new functionality.

AUTO SPEED DETECT (ASD)

This enhancement to PRIMOS provides the capability to automatically adjust current asynchronous communications hardware (AMLC, ICS1, ICS2) to operate at the speed of a terminal attached to a loginable line. The supported terminal speeds are 110, 300, 1200, 2400, 4800 and 9600 baud.

CONFIG Parameters for ASD Support

A terminal line must be explicitly set to ASD protocol for this enhancement to be operable. This is done by giving the AMLC ASD command from the supervisor terminal, normally at cold start of PRIMOS. The AMLC ASD command may be issued for a line with a logged in user but will not take effect until one of the conditions that cause a return to ASD protocol occurs.

In order for the speed detect algorithms to work, the following hardware configuration is necessary:

- The programmable clock speed must be set to 9600 baud. (This is the default speed and corresponds to a CONFIG word of X4XX.)
- The first jumperable speed must be set to 2400 baud. (This corresponds to the CONFIG word X5XX.)
- The second jumperable speed must be set to 4800 baud. (This corresponds to the CONFIG word X6XX.)
- The third jumperable speed (corresponding to the CONFIG word X7XX) is not used by ASD and may be set to any speed desired.

The AMLC requires modification to hardware jumpers to set these line speeds. The ICS1 and ICS2 controllers require that the following directive be issued:

```
ICS JUMPER 04540 11300 <ANY_NUMBER>
```

Without performing the jumpering modification or issuing the ICS JUMPER directive, only line speeds of 110, 300 and 1200 baud are supported. Attempted operation at other line speeds cause failure to establish correct line speed, resulting in loss of the use of that line. At some speeds, no line characters may be seen resulting in the ASD process waiting forever for the next character. The third numeric parameter for the ICS JUMPER directive must be given but it's value is not used by ASD processing.

The number of stop bits at each line speed is not modifiable by a command issued from the supervisor terminal. Simple modifications to the source module AMLDIM.PMA allow a field analyst or knowledgable customer to adjust the stop bits as necessary. The standard stop bit configuration for lines configured for ASD is two stop bits at 110 baud and one stop bit at all other speeds.

The AMLC ASD command is only applicable to current asynchronous implementations. Support is provided for the AMLC, ICS1 and ICS2 controllers.

The AMLC ASD command may only be issued for unassigned lines. The command is rejected if issued for an assigned line.

Any protocol can be used with lines configured for ASD. This is done by issuing the 'AMLC <protocol> line' command prior to the 'AMLC ASD line' command.

ASD operation on a line may be terminated by issuing any AMLC command for that line that does not specify ASD protocol.

There are a number of system wide configuration directives that affect the operation of ASD.

- The 'DISLOG YES' Configuration Directive should be given to ensure the return to ASD protocol if a user disconnects without logging out.
- The 'disctime' parameter of the AMLTIM configuration directive controls how often a user's line is checked for loss of carrier.
- The 'gracetime' parameter of the AMLTIM configuration directive controls the delay in return to ASD protocol for users not logged in.

> Caution
>
> It is very important that neither the 'disctime' nor the 'gracetime' parameters be set to zero. Setting either of these parameters to zero prevents recovery from certain error conditions and may result in loss of the use of ASD lines.

- The 'disctime' parameter must be at least twice as large as the 'ticks' parameter on systems using ICS1 or ICS2 controllers. If 'disctime' is set to a smaller number, disconnect checking will occur at every tick interval and gracetime processing will never occur. That is, gracetime processing occurs at all tick intervals that disctime processing does not occur.

AMLC ASD Command Syntax: The following are the commands to start and end ASD operations:

```
AMLC ASD line [CONFIG] [lword]

AMLC [protocol] line [CONFIG] [lword]
```

Refer to the System Administrator's Guide (DOC5037-2PA) for definitions of the parameters.

Note

If a protocol is not specified for an ASD line with a prior 'AMLC <protocol> line' command, the default TTY protocol will be enabled when speed detect processing has completed. If a prior AMLC command is issued for an ASD line but does not specify a protocol, TRANS protocol is enabled on the completion of speed detect processing. This is compatible with the current implementation of the AMLC command.

Compatability: This implementation of ASD is intended to operate with the AMLC hardware (both DMT and DMQ versions) and both ICS1 and ICS2 controllers using current software.

The user interface to a line in ASD mode can not be compatible with previous user interfaces due to the nature of ASD, that is, speed must be determined before any other interaction with a user terminal can take place.

Operation of ASD

A user attempting to login on a line that has been defined for ASD would repeatedly type (in upper case only) the character 'A' until the prompt 'Login please.' appeared on his terminal screen. The prompt would indicate:

- The terminal support software had detected the user's terminal speed
- The asynchronous hardware had been programmed to operate at that speed on the line to which the user was attached
- The line protocol had been changed from ASD to that line's normal protocol

If the user is running at a speed other than one of those supported for ASD or if the user types characters other than upper case A's, the line remains in ASD protocol. Character echo never occurs while in ASD protocol.

The users terminal must be configured to generate either 'mark' or 'odd' parity. The speed detect algorithms will not work with either 'space' or 'even' parity.

The line will be returned to ASD protocol under the following conditions:

- A local user's line returns to ASD protocol when the user logs out. Note that a local user performing login over login does not have his or her line returned to ASD protocol.

- A logged-through user's line returns to ASD protocol when the local host receives a 'call clearing' packet. This packet is sent from the remote host to the local host to indicate that the connection between them is being broken. This normally occurs when the logged-through user logs out from the remote host. This packet may also be received under certain error conditions that cause disruption of the link between the hosts. Note that a local user who had netlinked to a remote host and logged in there (Remote Login) does not have ASD enabled on logout from the remote system.

- Gracetime (see the AMLTIM Configuration Directive) has expired and the user has not successfully logged in.

- Carrier loss is detected for a non-logged in user. Logged in user's lines are returned to ASD by the logout on disconnect processing.

For users who are logging out, the message "Auto speed detect is enabled. Type "A"s until a login message appears." is printed on the user's terminal. The line's speed is then set to 1200 baud. Messages from external logout routines are garbled or not printed if the terminal speed is not also 1200 baud.

Conditions other than logout that cause a return to ASD protocol do not print a message at the user's terminal.

HALF-DUPLEX SUPPORT OVER MDLC

Dialup half duplex support for Prime-to-Prime links is now available over the MDLC only. This is functionally equivalent to and compatible with the half duplex protocol that was supported prior to Rev. 19.3 but was temporarily dropped.

PRIMOS CHANGES TO WARM START HANDLING

A new software interrupt that occurs at warmstart has been defined. The software interrupt causes all logged in processes to receive a condition WARMSTART$ signal. The default PRIMOS on-unit simply displays the following message and then returns:

```
***** WARM START *****
```

User software with ANY$ on-units should preferably ignore (continue to signal) this condition and let PRIMOS display the usual message and then resume normal execution. If the user ANY$ on-unit desires, it may deal with WARMSTART$ itself but it should, in all cases, display some message so the user is informed of the warmstart.

The visible effect of this change to warmstart is that logged out terminals no longer have the ***** WARM START ***** message displayed but phantoms now see this message (in COMO files) and logged in terminals see the message not only on the terminal but also in a COMO file if one is in use.

PRIMOS CHANGES TO SUPPORT THE PRIME/SNA PRODUCT

Other changes to PRIMOS for PRIME/SNA are largely internal and should have no effect on users not running PRIME/SNA. These changes include:

- ICS2 support has been added for synchronous lines for use by PRIME/SNA; this enhancement includes a new configuration directive to specify that the synchronous SDLC protocol should be down-line-loaded to the ICS2.
- Several ICS2 initialization error messages have been improved for clarity.
- Several new Ring 0 entrypoints have been added to PRIMOS to support PRIME/SNA and a new DTAR 0 segment has been added to contain the PRIME/SNA Ring 0 databases.
- Three new free storage classes have been added to the storage pools used by the communications software.

SOFTWARE PROBLEMS FIXED

The following non-reported bugs were fixed:

- The test to distinguish between normal status and marker status was reversed.
- The line number and loop counter were not being correctly incremented when a controller was skipped.
- An index register was not being initialized correctly in some cases in two loops.

DYNAMIC FILE UNITS

Dynamic File Unit allocation (DFU) has increased the number of file units within PRIMOS. The Rev. 19.3 limit of 3247 units per system and 127 per user are increased such that there are no internal system limits. To do this, file units are allocated dynamically from shared memory. The prior DFU limitation of 3247 file units is a reslut of the size of the Unit Tables Entries (UTE) that fit into one segment (64K words). DFU changed this limitation by using more than one segment

worth of storage and changed the file system routines to use double word pointers to access multiple segments.

DFU has not changed the basic unit table structures but the FORTRAN modules have been recoded to take advantage of the pointers in PLP and use the Dynamic Storage Allocators to allocate unit tables. Each user has a single pointer in his PUDCOM area that points to the start of the unit table (UTBL). There is a unique unit table for each user process.

CHAP COMMAND ENHANCEMENTS

Functional Overview

The new scheduler enhancements add two special purpose priority levels to the current scheduler. The first, called IDLE level, defines a class of users that use spare system resources. Processes relegated to the IDLE priority level run only when no other process at any other priority is eligible to run, that is, when the system is idle. IDLE processes comprise a queue that is conceptually below the low priority queues. The second new priority level, called SUSPEND level, defines a class of users that are temporarily blocked from receiving service. Processes at SUSPEND priority level are ineligible for service for an indefinite period of time during which they are not subject to the system idle time limit. This facility may be used by the System Administrator to speed very high priority jobs through a loaded system or to synchronize the execution of large jobs that thwart the system by thrashing against each other.

Movement between existing priority levels and the two new levels is controlled by the CHAP command. Both the supervisor version and the normal user versions of the command have been modified to accommodate the new levels. The new form of the supervisor command may be employed to place users on either the IDLE or SUSPEND queue. Processes remain idle or suspended until they are explicitly placed back on one of the existing priority levels (0-3) by CHAP. The new form of the user command allows a user process to CHAP itself into the IDLE level only and to return itself to the default user priority level.

In addition, the routine SPAWN$ is enhanced to allow phantoms to be spawned directly into the IDLE priority level. This involves implementing the priority argument to SPAWN$, which currently exists but is ignored. A privileged process is allowed to spawn phantoms into any priority level except SUSPEND.

CHAP Command Syntax

Supervisor Command: This section presents the revised syntax of the supervisor version of the CHAP command and describes the new options in detail. Refer to the System Operator's Guide for a discussion of the existing CHAP parameters.

The new form of the CHAP command is:

```
CHAP  { -userno } { -IDLE                    }
      { ALL     } { -SUSPEND                 }
                  { priority  [timeslice]    }
```

userno is the target process' user number. If ALL is specified, all processes except certain privileged or special purpose processes; such as user 1, NETMAN, and slaves, are affected.

IDLE places the specified process at IDLE priority level. SUSPEND places it at SUSPEND level. (Refer to the previous section for a description of the IDLE and SUSPEND priority levels). The new priority level remains in effect until it is explicitly changed again.

IDLE may not be specified if the target process is a terminal user; an attempt to do so results in an error message. When a terminal process is suspended, a warning message is issued. Any characters typed before the process is made eligible again may be lost. Neither IDLE nor SUSPEND is valid for time-critical processes, which have their timeslices set to -1. The timeslice must be reset to a normal value before the process can be made idle or suspended. IDLE and SUSPEND are only valid for normal terminal users and phantoms. If the requested priority change is not valid for the specified user when the -userno argument is supplied, an error message is issued. If the ALL option is specified, processes for which the priority change is invalid are simply skipped and a warning message giving the number of processes that were not affected by the command is issued. This form of the CHAP command can only be issued from the supervisor terminal.

User Command: The syntax of the user version of the CHAP command is:

```
        ⎧ UP      ⎫
        ⎪ DOWN    ⎪
CHAP    ⎨ LOWER   ⎬ [timeslice]
        ⎪ IDLE    ⎪
        ⎩ DEFAULT ⎭
```

UP moves the user's priority up one level but no higher than the current default user priority level (The default user priority level may be reset by the system operator through the supervisor terminal version of the CHAP command). The UP option has no effect if the process is at IDLE level.

DOWN moves the user's priority down one level but no lower than the lowest normal priority level configured (usually 0). The DOWN option can not be used to move the process into the IDLE or SUSPEND levels. CHAP DOWN is equivalent to CHAP LOWER 1 (see below).

LOWER lowers the user's priority by nnn levels, but no lower than the lowest normal priority level configured (usually 0). nnn must be an integer value in the range 1 through 7. timeslice is optional; it specifies a new timeslice value in octal in units of tenths of a second. The new timeslice value must be smaller than the users's current timeslice value; in other words, the user may shorten his timeslice but may not lengthen it. The LOWER option can not be used to move the process into IDLE or SUSPEND level.

IDLE places the user at IDLE priority level. This option may only be issued from a phantom. Terminal users may not make themselves IDLE.

DEFAULT resets the user to the current default user priority The default user priority level may be reset by the system operator through the supervisor terminal version of the CHAP command.

SPAWN$ Interface Changes: The priority argument to SPAWN$ is an element of a structure describing various characteristics of the new process. Currently, the priority field is ignored. A phantom is assigned a priority of 1 if the spawner is User 1; otherwise it is assigned the priority of the spawning process. SPAWN$ allows phantoms to be spawned into any priority level except SUSPEND. This is implemented through the priority argument as follows:

Value of Priority	Meaning
-2	Place phantom at IDLE level
-1	If the caller is user 1, assign the phantom to priority level CURLEV (currently 1). Otherwise, assign the phantom to the spawner's priority level
0-3	Assign the phantom to priority level 0-3 or the spawner's priority level, whichever is lower.

All other priority values are invalid and result in an E$BPAR error code being returned from SPAWN$. Note that the default action (taken when the priority is -1) is different for user 1 than it is for other users.

BATCH

NEW FEATURES AND CHANGES

Batch EPF Support

- The Batch subsystem was modified to save the EPF attributes of the process submitting a request and passing them on to the actual job.
- The size of the batch queue entry has been expanded.

Prior to Rev. 19.4 Batch, there was no way to limit the EPF attributes of a Batch job. These EPF attributes include the maximum number of private static and dynamic segments the process is permitted, as well as the maximum number of cammand levels and program invocations. These limits are set when a terminal process logs in. For phantoms, however, they are copied from the process which creates the phantom. This means that every Batch job would inherit the same EPF attributes as the Batch monitor which creates all Batch phantoms. Therefore, Batch was modified so that the EPF attributes are acquired from the process submitting a Batch job and passed on to the Batch phantom when the job is run.

Since there is a delay between the time a Batch job is submitted and the time it is run, the new EPF attributes must be stored in the Batch job queue entry. This resulted in a change in the overall length of each Batch job queue entry. Since the old entry lengths are incompatible with the new entry lengths, all Batch jobs submitted prior to Rev. 19.4 Batch must be either executed or deleted before upgrading to Rev. 19.4 Batch. A new error code has been added to Batch which checks through all active Batch job queues to see if there are any queues with active job entries of the old length. If there are, a message is sent to the supervisor terminal informing the operator that the Batch queues are not completely Rev. 19.4 compatible.

Note

Rev. 19.3 Batch will continue to function with Rev. 19.4 Primos. However, there is no Batch support for EPFs prior to Rev. 19.4 Batch.

Longer CPL Arguments

Prior to Rev. 19.4 Batch, the length of the argument to be passed to a CPL Batch job was limited to 80 characters. With Rev. 19.4 Batch, this limit has been doubled to 160 characters.

Batch Support for Dynamic File Units

Prior to Rev. 19.4, a user could specify that a Batch job open its input file only on units less than 126. With Rev. 19.4 Batch, this restriction has been removed.

SOFTWARE PROBLEMS FIXED

Open CPL Files: A major problem fixed at Rev. 19.4 is the situation where queues hang for no apparent reason. The problem was that CPL jobs have always kept the temporary file they use for command input open until termination of the job. COMI jobs, on the other hand, closed the temporary file before the job terminated. This led to the occasional situation where a CPL job from one queue used the same temporary filename as a COMI job from a second job queue. When the COMI job finished it would try to delete the temporary file. However, this file was already held open by the CPL job. Therefore, the COMI job would have to wait until the CPL job terminated before it could finish. This effectively hung the queue where the COMI job had been submitted.

Monitor Attaches to BATCHQ Directory: Another problem fixed at Rev. 19.4 is that of the BATDEF file not being found even though it existed in the BATCHQ directory. The problem was that the Batch monitor was not always attached to the right directory before trying to open the BATDEF file.

The Batch monitor now does an explicit attach to the BATCHQ directory before opening the BATDEF file (SPAR 3002389).

File Units and Monitor Waiting: File unit handling was modified to support dynamic file units up to 128. Also, to avoid hanging queues, COMI jobs no longer close their temporary files until the job is finished.

On a heavily loaded system, there are certain situations where race conditions develop between the Batch monitor and Batch jobs. The length of time that the Batch monitor waits for a Batch job to reach a certain stage in its processing was not always sufficient. With Rev. 19.4 Batch, the length of time that the monitor waits has been increased. This should provide a more stable and more reliable Batch product (SPAR 2004578).

BOOT_CREATE

NEW FEATURES AND CHANGES

None

SOFTWARE PROBLEMS FIXED

Password Prompt: The option -NOPASS has been replaced by the option -NO_QUERY (-NQ). This option prevents any prompts for passwords being given during a BOOT_CREATE session (SPAR 3003495).

Access Rights: If SEGSAM or SEGDAM type files were explicitly named in the list file for BOOT_CREATE, then in attempting to check the access, error conditions were always raised. This was due to BOOT_CREATE attempting to open each sub-entry of the SEGDIR to check its access rights. BOOT_CREATE now checks access rights by opening only the top level of the SEGDIR (SPAR 3007629).

Correction to List File: Files of the type SEGSAM were not supported, when explicitly named, in the list file of BOOT_CREATE. The file type was isolated as an attribute and checked against a list of supported filetypes. The list held a name SEGAM instead of SEGSAM (SPAR 3007630).

Options: If multiple options are given for one of the files in the list file of BOOT_CREATE, then BOOT_CREATE tells the user that it will default to the -YES option instead. It then goes on to test that the -CHECK option has been used and takes -CHECK in preference to -YES (SPAR 3007631).

If multiple options have been used, BOOT_CREATE uses -YES by default.

The -CHECK Option: The -CHECK option does not present blank lines to MAGSAV. This prevents the "Syntax error" message from being displayed for each blank line. -CHECK still has the rest of its previous functionality (SPAR 3008138).

BRMS

NEW FEATURES AND CHANGES

At Rev. 19.4, BRMS attempts to replace any in-use EPF's encountered during restore operations (archive_restore, transport_restore and backup_restore). The functionality is similar to that of COPY and involves restoring the EPF into a temporary file and then replacing the target EPF. The old file is renamed from name.RUN to name.RPn, where n is a digit.

CMPF/MRGF

NEW FEATURES AND CHANGES

CMPF was fixed so that it does not clear the last digit of control characters.

CONCAT

SOFTWARE PROBLEMS FIXED

Missing Keyword: -If NEW_TITLE is not specified with the TITLE subcommand, the user receives the message 'Missing keyword option(cmdl$a)' (SPAR 3002103).

COPY

NEW FEATURES AND CHANGES

At Rev. 19.4, the COPY command, used on EPF files, attempts to remove a mapped EPF from a user's address space before performing the COPY operation. This is done to ensure that the EPF is removed from the disk and from the user's address space.

The COPY command also allows the copying of an in-use program EPF or EPF library. The suffix on the command (.RUN) is incremented to the next revision (RP0) to allow the old version of the EPF to continue to function. The newer version of the EPF is then used and the older version can be deleted either when the EPF is removed from the user's address space or the user logs out. There may be up to nine versions of an EPF with suffixes from .RP0 to .RP9.

COPY_DISK

NEW FEATURES AND CHANGES

COPY_DISK messages have been converted to upper/lower case.

The routine YESNO, which obtains yes or no answers from the current input stream, has been modified to be case insensitive. The acceptable responses are (in either upper or lower case) YES, Y, YE, NO, and N.

DELETE

NEW FEATURES AND CHANGES

At Rev. 19.4, the DELETE command used on EPF files attempts to remove a mapped EPF from a user's address space before performing the DELETE operation. This is done to assure that the EPF is removed from the disk and from the user's address space.

ED

SOFTWARE PROBLEMS FIXED

ACCESS_VIOLATION$ error message was raised when issuing the command MODE L. This problem has been fixed (SPAR 3009488).

The problem with using MOV STRA INLIN or MOV STRA /FOO/ has been corrected (SPAR 3009542). This SPAR duplicates SPAR 3007868.

EDB

NEW FEATURES AND CHANGES

EDB now displays up to 32 characters in entry names so that users can see the part of the names that BIND is dealing with. No method exists to display only 8 or 6 characters. Since 32 characters take up a large portion of the screen, only two names are output per line.

PERMANENT RESTRICTIONS

EDB does not search binary files for the .BIN suffix; it must be supplied by the user.

EDIT_PROFILE

NEW FEATURES AND CHANGES

Four new user, or project, and system default attributes, and new conversion procedures were added for Rev. 19.4.

The LIST_SYSTEM command has been modified to display system default attributes.

SOFTWARE PROBLEMS FIXED

The -DEFAULT option of EDIT_PROFILE now works correctly (SPAR 3002167).

EDIT_PROFILE no longer accepts null user ids or null project ids (SPAR 3003324/3002290).

EDIT_PROFILE no longer accepts invalid default project names. If the user enters an invalid default project name, EDIT_PROFILE returns an error message followed by the EDIT_PROFILE prompt. The rebuild now runs smoothly (SPAR 3004176).

In initialization mode, if the user responds to the question 'PROJECTED NUMBER OF USERS' with a number larger than the size that EDIT_PROFILE can handle, EDIT_PROFILE responds with the error message "SIZE MUST BE A NUMBER BETWEEN ZERO AND 28004" (SPAR 3005232).

When changing a user's initial attach point, EDIT_PROFILE now allows the user to enter 'none' in the retry loop (SPAR 3004845).

Quits are now inhibited during the ADD_PROJECT command (SPAR 3003327).

If the user shuts down the partition, from the supervisor terminal, that he or she is attached to and then enters EDIT_PROFILE, the user now receives just one error message when exiting from EDIT_PROFILE using QUIT (SPAR 3002595).

If EDIT_PROFILE is aborted during the entry of a project and then a list of projects is requested, EDIT_PROFILE no longer terminates with the message "END OF FILE. PVF HEADER" (SPAR 3001396/3003327).

EMACS

NEW FEATURES AND CHANGES

The following summarizes the new features and changes.

- New EMACS functions
- New EMACS Atoms
- New terminal support: the Prime PT200
  - Command line arguments -TTP PT200 and -TTP PT200W
  - Saving/Restoring the state of PT200 Screen Display Mode
  - New command line option: -SAVE_SCREEN for PT200 in 80x48 mode
  - Change in the setup of the PST100 CURSOR-FUNCTION/NUMBER Pad in EMACS Fundamental Mode and its relation to the PT200 PF/NUMBER Pad
- Changes to EMACS which are not terminal-specific
  - Removal of the limitations on the PRIMOS_INTERNAL_SCREEN command
  - New global switch controlling level of user feedback: VERBOSITY_LEVEL$
  - OPTIONAL change to feedback from quit command when buffers are unsaved
  - OPTIONAL change to feedback from <CTRL>P
  - Slight change to internals of COBOL mode
  - New Command or macro: fundamental for reestablishing Fundamental Mode bindings
- Changes to the Standard User Interface (SUI)
  - Change to PRIMOS_COMMAND command
  - Substitution of the FIND_FILE for the GET_FILE function
  - Addition of new predefined PRIMOS commands: LD, TLD, VLD
  - Changes to attach and spool commands
  - Miscellaneous changes to internals of the SUI

- Horizontal scrolling has been added to the SUI
- Changes to the SUI HELP screens
- SUI support in both 80x24 and 132x27 display modes for the PT200 terminal
- Normal operation of SUI/SUIX on PT200 terminal
- A note on the BS and DEL keypaths in SUI/SUIX versus Fundamental Modes

- Changes in the EMACS* directory
  - Contents of EMACS*
  - Contents of EMACS*>INFO
  - Contents of EMACS*>EXTENSIONS
  - Contents of EMACS*>EXTENSIONS>SOURCES
  - Contents of EMACS*>EXTENSIONS>SUI
  - Contents of EMACS*>EXTENSIONS>SUI>SOURCES
  - Contents of EMACS*>EXTENSIONS>SUI>UNSHARED

Caution

Unless your user process has been allocated an adequate number of dynamic segments at Rev. 19.4, you will be plagued by EMACS telling you 'NOT ENOUGH SEGMENTS' and having many of your commands aborted. To run EMACS without getting such errors, you should have your System Administrator set your allowed number of dynamic segments to at least 100. This should only be done by the System Administrator using EDIT_PROFILE.

New EMACS Functions

The following previously-undocumented functions are documented here in a format similar to that used in EMACS*>INFO>DESCRIBE_EMACS where functions can be accessed via the describe command.

autoload_lib Extended command that fasloads a file and executes the given command.

(autoload_lib atom string)

The two required arguments are an atom which is the command to be executed and a string which is the pathname of the file containing the command. The file must be in "fasload" format. (See fasdump, fasload, and dump_file.)

clerical$ This is another user type similar to programmer$ (see EMACS Reference Guide (IDR5026)) but it is much simpler. All files are loaded with fill mode on.

create_text_save_buffer$
Goes to or creates the next buffer in a circular list of ten buffers, deletes the contents of that buffer, and inserts the string argument into that buffer. It saves the text in a temporary buffer.

(create_text_save_buffer$ string)

where the argument is a string of text to be saved.

cursor_info Returns the asked for property of a cursor.

(cursor_info cursor prop {new_value})

Valid properties are buffer_name, line_num, char_pos, and sticky. new_value is only valid for sticky and is ignored for the other properties. If an illegal property is asked for, the string $ERROR$ is returned and an informational message appears.

decimal_rep Same as integer_to_string

def_auto Creates a function object stub before the function is loaded. This makes functions available with their description strings without actually loading the function. When the function is invoked, then it is actually loaded. This was used prior to Rev. 19.2 to create the libraries to be loaded. It may be useful for loading personal libraries.

(def_auto atom string string)

atom is the name by which the function is to be invoked, the first string is the help description string, and the second string is the pathname of the file containing the function.

found_file_hook
This is the function that checks the suffix when a new file is brought into a buffer and then turns on the mode associated with that suffix. (See EMACS Reference Guide (IDR5026), Appendex A.)

get_cursor Returns a list of the cursor properties.

```
(get_cursor {cursor})
```

The list is of the form ("buffer_name" line_num char_pos sticky); for example, ("main" 1 1 true)

go_to_buffer Like select_buf but returns a cursor to the first line of the buffer and positions the user at the first line.

```
(go_to_buffer "buffer_name")
```

ld Does a PRIMOS LD command at the current attach point. (Operative in SUI only)

load_lib Fancy fasload with message and bit to tell if the load was successful.

make_cursor Returns a cursor generated from the argument.

```
(make_cursor buf_name lnum cpos {sticky})
```

The sticky bit defaults to true if not specified.

nthcar Returns the nth car of the list.

```
(nthcar list number)
```

The car to return is specified by the number.

pending_reenter
Same as suppress_redisplay.

primos_smsgl Send a PRIMOS message.

```
(primos_smsgl any string)
```

any is either the addressee's name or process number and string is the message to be sent.

programmer$ This is a user type. It sets the mode of the buffer depending on the suffix of the file name read into the buffer. (See EMACS Reference Guide (IDR5026) under Changing File Hooks.)

reset_tabs SUI tab function to set the tabs to every five spaces up to column 130.

rest$ Same as suffix$.

restrict_to_sui$
Bound to functions appropriate only when using the SUI. Further documentation below.

scan_errors (scan_errors language) Special function which scans the current buffer for errors in the format for language. language is a non-quoted atom. The list of languages currently is: C, RPG, FIN, PMA and TSI. TSI refers to all of Prime's common envelope compilers; PL/1, PL/1G, Pascal, CBL, VRPG, and SPL.

search_back_first_charset_line
Same as search_bk_in_line.

search_back_first_not_charset_line
Same as verify_bk_in_line.

search_charset_backward
Same as verify_bk.

search_charset_forward
Same as forward_search.

search_for_first_charset_line
Same as search_fd_in_line.

search_for_first_not_charset_line
Same as verify_fd_in_line.

search_not_charset_backward
Same as verify_bk.

search_not_charset_forward
Same as verify_fd.

share_library$

(share_library$ pathname)

Shares an efasl file. It is used in init_emacs when sharing EMACS. The shared efasl file must have only defun's and defcom's in it.

show_lib_alc$ Prints at the terminal what the current shared EMACS library segments are. It is used in init_emacs when sharing EMACS.

sublist (sublist list number {how_many}) Sublist is a list equivalent to substr. It returns the list starting at the numbered car. If the optional how_many is specified, it returns the list with that number of items starting at the numbered car.

suffix$ returns the substring after the rightmost '.' in the name of the current buffer or the passed treename. If the string has no periods or there is a '>' after the rightmost period, this returns a string consisting of a single space.

(suffix$ [string])

string is an optional argument and is assumed to be a pathname.

sui_exchange_mark
SUI version of exchange_mark command. More user feedback is provided.

sui_horiz_left
SUI version of horiz_left command.

sui_primos_command
SUI version of primos_command command.

sui_set_tabs Establishes TAB settings for SUI users.

tld Does a PRIMOS LD -LONG -SRTD command at the current attach point. (Operative in SUI only.)

unused_key_feedback$
Bound to unused keys in SUI.

unused_mb_key_feedback$
In mb_mode, bound to unused keys in SUI.

vld Does a PRIMOS LD -LONG -SRTN command at the current attach point. (Operative in SUI only.)

New EMACS Atoms

verbosity_level$
Controls level of user feedback from quit function and CONTROL-P Handler. The default value is 2. Further documentation below.

sui_info_message_time$
Controls length of time that certain SUI error messages are displayed at the bottom of the screen. The initial value is 3000, measured in milliseconds. Further documentation below.

A new COMPILE command has been installed that is much faster than the previous one in scanning and displaying errors. It allows a user to include language options on the COMPILE command line. Languages supported are C, COBOL, FTN, PMA, RPG, PL/1, PL/1G, Pascal, CBL, VRPG, and SPL.

New Terminal Support

The Prime PT200 is now supported.

Command Line Arguments

EMACS now recognizes two new arguments to the -TTP Command Line option:

```
-TTP PT200
-TTP PT200W
```

When given with no additional -WIDTH or -HEIGHT qualifications, -TTP PT200 enters EMACS with normal support for the PT200 terminal. (See below for details on SUI support.) EMACS assumes you wanted its screen defaults for the PT200, which are -WIDTH 80 and -HEIGHT 24.

If you enter EMACS with -TTP PT200W, EMACS comes up in 132x27 display mode. 132 character-wide files can be displayed with no need for horizontal scrolling. The text display area is 24 lines high with the normal 3 lines for mode line and minibuffer.

Normal use of -HEIGHT and -WIDTH Command Line Options: If you add only the option -WIDTH N to the EMACS command line and N is less than 81, EMACS starts up after having set the PT200 terminal into 80x24 display mode. If N is greater than 80 and less than 133, EMACS starts up on the PT200 terminal after having set the terminal's display into 132x27 mode. In both cases, the actual EMACS display window is N characters wide, which may be smaller than 132 characters.

If you add only the option -HEIGHT N to the EMACS command line, and N is greater than 5 and less than 25, EMACS starts up in normal 80x24 display mode. If N is between 25 and 27, and -WIDTH is greater than 80, EMACS starts up after having placed the PT200 in 132x27 mode. In both cases, the EMACS display window is N lines high, including mode line and EMACS MiniBuffer lines.

Thus the -TTP PT200W Command Line Option is simple shorthand for the text:

```
-TTP PT200 -HEIGHT 27 -WIDTH 132
```

on the command line. In fact, if your command line contains -TTP PT200W, any arguments to -HEIGHT or -WIDTH are ignored. If you want any nonstandard screen height or width, use -TTP PT200 and go from there.

Limitation - No PT200 Display Mode Switching: It is not possible at this release to issue any EMACS command that switches a given EMACS session between the PT200's 132-column and 80-column modes. To switch display modes, the user must exit EMACS and return in the other mode.

Saving/Restoring the State of PT200 Screen Display Mode: The PT200 can operate in any of four configurations of its own internal display memory: 80x24, 80x48, 132x27, and 160x24.

When EMACS is in control of the terminal, it is either in 80x24 mode or in 132x27 mode. This is because EMACS makes no use of within-terminal display scrolling capabilities, either horizontal or vertical. EMACS handles all of the scrolling responsibility itself. When you are not using the SUI, the total size of your EMACS screen may be between 6 and 27 lines in height and between 10 and 132 columns in width.

The user can call EMACS from the PT200 terminal at PRIMOS level when it is operating in any of the four configurations of its internal display memory.

In all cases, upon normal exit from EMACS, the PT200 is returned to the going-in state of its internal display memory.

New Command Line Option: -save_screen for PT200 in 80x48 Mode: The normal behavior of EMACS upon the termination of an EMACS session is to clear the contents of the terminal display memory and leave the user with the PRIMOS prompt at the top of the screen.

Users are now able to invoke EMACS with the addition of the new Command Line Option -save_screen. This does not have any effect on any terminal other than the PT200 (and its Prime terminal successors).

The -save_screen option has an effect only when the PT200 terminal is in 80x48 mode before going into EMACS and the 132x27 mode is not used inside of EMACS. The save_screen option is accepted without error but has no effect if the user invokes EMACS when the PT200 terminal is in any state other than 80x48 screen display.

When the above conditions are satisfied at EMACS invocation, EMACS asks the terminal to remember the current location of the cursor within its display memory. Upon exit from EMACS's control, EMACS restores the PT200 display to its going-in state and then, if that state was 80x48 and -save_screen was given, EMACS instructs the terminal to restore the cursor to where it was at the start of the session (unless the user executes a primos_internal_screen command during that EMACS session, as will be documented further).

The result of this new feature is that the second page of display memory (if any) is saved and restored after the EMACS session is completed. Since EMACS does all its work in the first page of display memory when it places the terminal in 80x24 mode during a normal session, nothing on the second terminal page is touched during the EMACS session (assuming the user does not give the primos_internal_screen command). Therefore, upon exit from EMACS, the user sees the PT200 screen as it appeared just as EMACS was being invoked. The few lines above the EMACS invocation are still there on the screen. If the user scrolls the PT200 back to the first page of display memory, the page will be blank. EMACS was doing all of its work in that page and EMACS clears it prior to exit.

Relation Between the -save_screen EMACS Command Line Option and the primos_internal_screen EMACS Command: This restoration of cursor positioning outside of EMACS has a considerable benefit for users of the primos_internal_screen command.

When the primos_internal_screen command is operating, EMACS normally restores the PT200 display mode to its going-in state and only then issues the PRIMOS command the user entered. (This command can now be external as discussed later.) Just before the beginning of the execution of the desired command, therefore, the screen has the appearance it did when EMACS was invoked. In fact, the screen appears as it would have at the conclusion of the EMACS session, had the primos_internal_screen command not been given.

An EMACS user with -save_screen in operation sees the display and cursor restored to their going-in state prior to the execution of the internal PRIMOS command. The user sees the last few command lines given at PRIMOS level prior to their invocation of EMACS. Internal PRIMOS commands that result in output lines cause the original EMACS invocation command to scroll upward, possibly off the screen.

Successive operations of the primos_internal_screen command produce output at the bottom of this second page of display memory. Thus, PT200 user context can be saved in successive primos_internal_screen commands.

Upon exit from EMACS, the -save_screen user retains the last 24 lines of display memory even if these were executed from inside EMACS via the primos_internal_screen command.

Limitations on the Usefulness of -save_screen: The -save_screen command line option is of benefit to the user if (and only if) the PT200 cursor was somewhere in the second page of display memory when EMACS was invoked. Because the cursor is at the bottom of display memory for the great majority of times that EMACS is called, the benefit of context-saving is obtained most of the time.

-save_screen or -ss Command Line Abbreviation: To save typein, the EMACS command line abbreviation -ss may be used to obtain this feature. A PT200 user should keep in mind that, to get the benefit of the -save_screen functionality, the user must add it to the PRIMOS command line (or abbreviation) used to invoke EMACS; -ss or -save_screen are not defaults.

Change in the Setup of the PST100 CURSOR-FUNCTION/NUMBER Pad in EMACS Fundamental Mode

To avoid disrupting current PST100 users' EMACS Fundamental Mode interfaces at this release, the default handling of the PST100 PF/N pad has not been changed. For Fundamental Mode, the PF/N pad setting is not affected at startup. SUI/SUIX users' PF/N pads are, however, forced into PF mode at startup. The PST100 commands which force the PF/N Pad into one or the other mode have also been removed from the respective refresh functions used in both Modes.

Thus, the EMACS Fundamental Mode user wishing to bind functions onto the PF/N pad can easily include in their library file the following EMACS command which switches the pad into PF mode.

```
(send_raw_string "~c[[>10h")
```

The command stays in effect for the balance of the session unless countermanded by a similar raw string terminating in "10l" (the final character is a lowercase L). The PST100 (refresh) function does not affect the mode setting of the PF/N pad.

The PST100 SUIX user gets the switch into PF mode in the initialization sequence used for that mode. The PST100 (sui_refresh) function does not affect the mode setting of the PF/N pad.

Cautions on the Use of the PST100 PF/N Pad with the primos_internal_screen Command

One major but subtle exception to this policy is important for SUI/SUIX users only in the control sequences given to the PST100 when returning from the primos_internal_screen command. (This command is accessible either by name or by prefixing !! to the argument of a primos_command command. It is also used in the STAT, LD, TLD, and VLD commands noted elsewhere in this document.) When EMACS is repainting the SUI/SUIX screen, the control sequence forcing the PF/N pad back into PF mode is invariably given. Thus, if the SUI/SUIX user for some reason had the PF/N pad in N mode prior to such a command, the pad would be back in PF mode after it.

The PF/N pads of Fundamental Mode users of the PST100 terminal are not affected on returning from these commands but they are affected upon exit to PRIMOS level. If the user starts EMACS Fundamental Mode with the PF/N pad in N mode, it remains in N mode inside EMACS. If the pad is switched to PF mode within the session, it remains so through normal refresh functions. However, if a primos_internal_screen command is given, the terminal is reset to its going-in state, including N mode for the PF/N pad, for that command and it is not forced back into PF mode upon return to EMACS Fundamental Mode.

The CURSOR-FUNCTION/NUMBER PAD (CF/N) on the PT200

The pad corresponding to the PST100's PF/NUMBER pad is the CURSOR-FUNCTION/NUMBER (CF/N) pad. (The old PF keys have moved elsewhere on the PT200 keyboard.) The CF/N pad can be toggled between these two modes from PRIMOS level, using the NUM LOCK key. When in NUMBER mode, the NUM LOCK LED is lit. When in CURSOR-FUNCTION mode, the pad sends several different ESC sequences to the host, which EMACS SUI uses for cursor control and the like.

Behavior of PT200's CF/N Pad in EMACS: At this release of EMACS for the PT200 terminal, the default behavior of the CF/N pad is similar to the PF/N pad of the PST100. It is placed in CURSOR-FUNCTION mode as SUI/SUIX modes are started and is untouched in the startup of Fundamental Mode. This facilitates users' binding their own functions to the CURSOR-FUNCTION pad in both SUI/SUIX and Fundamental Modes. Under default conditions for PT200 Fundamental Mode users, the coming-in value of their PF/N pad switch determines whether it is possible to type in numbers on the CF/N pad. If it came in in number mode, that's what it produces inside EMACS. If it came in in CF mode, it produces Invalid command: feedback, and occasional inserted characters, from CF keys not bound to anything useful.

How to Get the PT200 CF/N Pad to Function as a NUMBER PAD in Fundamental or SUI/SUIX Modes: If you are a Fundamental Mode user who wishes to use the CF/N pad in NUMBER mode, simply cause the execution of the following PL codes:

(send_raw_string "~c[[>10l")	to turn number mode ON. The last character in the command string above is a lowercase L
(send_raw_string "~c[[>10h")	to turn number mode OFF

You can bind functions which generate the above raw strings to the PT200 function key(s) of your choice.

Note that the PT200's NUM LOCK key, for SUI/SUIX modes, is bound to functions that do one of two things. If the CF/N pad is in CF mode, the key runs the DELETE FORWARD function, for either characters or words, depending on the CHAR/WORD switch. If the pad is in Number mode (with the NUM LOCK's LED on), it runs a function that turns CF mode on and turns off the LED.

See above for a discussion of the effects of the primos_internal_screen command's effect on the current setting of the PT200's CF/N pad. What applies to the PST100's PF/N pad applies equally to the PT200 terminal's CF/N pad.

Disposition of Unused PT200 CF/N Pad Keys in SUI/SUIX and Fundamental Modes: All keys on the CF/N pad which are not bound to useful functions have been bound, for SUI and SUIX users only, to a simple EMACS function that echoes 'This key is not used' in the minibuffer for a fixed 1 second of time-out, but which does not ring the terminal's bell. This function is bound to all members of CF/N pad when the SUI/SUIX user presses them with the SHIFT, CONTROL, or both SHIFT and CONTROL keys held down.

This is not true for Fundamental Mode on the PT200 terminal. This means that all keys on the CF/N pad, in all augmentations, produce the 'Undefined key' EMACS feedback if pressed in Fundamental Mode with the CF/N pad in CF mode.

CHANGES TO EMACS WHICH ARE NOT TERMINAL-SPECIFIC

Removal of the Limitations on the PRIMOS_INTERNAL_SCREEN Command

It is now no longer necessary to restrict the commands given via the primos_internal_screen command to internal commands. Since EMACS is now running as a shared library, there are no longer any conflicts with other software. The only exception to this rule is that you still can not invoke a second EMACS from within the surrounding EMACS. However,

you can run internal commands, NETLINK, compilers, and so forth, via the primos_internal_screen command, interact with your screen as if you were running the subordinate software at the PRIMOS level, and return at the conclusion with your full EMACS editing context undisturbed.

New Global Switch Controlling Level of User Feedback, VERBOSITY_LEVEL$

At this revision of EMACS, a new global variable, verbosity_level$, has been added to control some of the behavior of the EMACS user interface. The value of this variable is initially used only in controlling the user feedback given by the QUIT command and by the handler for <CTRL>P, documented below. Command writers are free to write macros that consult this variable and make their user feedback contingent on its value or to change the value of the variable itself, thus controlling the behavior of commands using it. Of course, if they do change it, they must take care to make its new value a valid one.

There are four logical levels for the new verbosity_level$ variable. They are all entered as integers:

0 This signifies the old-style feedback or the lowest possible level of user feedback for newly-written functions.

1 This signifies a low level of feedback.

2 This signifies a medium level of feedback and is the new default value for the verbosity_level$ global variable.

3 This signifies a high level of feedback and, while it is not used at present, it could be used in the future.

It is the intent that EMACS command writers test the value of this variable to do particular actions based on the value of verbosity_level$.

With reference to the two commands below, if users wish to reinstate the original behavior of the QUIT command or of the response to the <CTRL>P key, all they have to do is incorporate the following into their user library file:

```
(setq verbosity_level$ 0)        where the argument is an unquoted
                                 zero
```

As usual, users can write macros that control the value of verbosity_level$ in a more automated, user-friendly manner.

Optional Change to Feedback From QUIT Command When Buffers are Unsaved

One of the more cryptic pieces of user feedback that EMACS used to produce was given if the user attempted to quit EMACS via the <CTRL>X<CTRL>C command in Fundamental Mode or by the key labeled EXIT (PT45) or QUIT EMACS (PST100/PT200) in the SUI mode.

If there are no buffers with unsaved changes at the time of this command, EMACS simply halts with no errors and no problems.

However, if there are buffers with unsaved changes, that is, there are buffers for which the * is shown on the mode line between buffer name and the associated pathname, a question appears in the Minibuffer. The language of the question has been changed. The new default behavior of EMACS is to prompt:

> Above list of modified buffer(s) not saved to file(s). Quit anyway?:

If the user wishes to retain the old form of the question in this case, the only thing required is to set verbosity_level$ back to 0.

Optional Change to Feedback From <CTRL>P

At previous EMACS releases, typing <CTRL>P produced clearing of the screen and display of the text:

> Type REN to abort command, START to continue.

The newly added default version of EMACS's feedback in this case makes the distinction between REN and START much clearer and also prevents the unintentional erasing of an entire EMACS editing context.

The first thing that happens is that the terminal's bell is rung and the EMACS screen is cleared. All typeahead is flushed and the terminal display memory is restored to its going-in state. Then the following text appears on the screen.

> CONTROL-P typed.
>
> To really Quit from EMACS, type Q
> To return to EMACS and Abort the current command (if any), type A
> To return to EMACS and Continue with no interruption, type C
>
> Confirm your choice with the RETURN key.
> Typing RETURN without making a choice is the same as Continue.

(If no EMACS command was executing when you pressed CONTROL-P, then Abort is the same as Continue.)

You may need to refresh the screen if you choose to re-enter EMACS.

C, A, or Q:

All responses other than a RETURN or Q, A, or C preceding a RETURN, result in the repetition of the last prompt line.

Thus, if <CTRL>P is mistakenly typed, the user does not lose the entire editing context and return to PRIMOS level without a hearing a bell, getting the typeahead flushed, and then typing Q and then RETURN. The distinction between returning after Aborting an ongoing EMACS command, and simply Continuing, is also somewhat clarified.

Note

The user is told to refresh the screen after returning control to EMACS because it will probably be necessary. SUI users need to press <CTRL>L or their CLEAR or RESET keys to reestablish the SUI's reverse-videoed area at the bottom of the screen.

If the user wishes to retain the old form of the dialog, the only thing required is to set verbosity_level$ back to 0.

Change to Internals of COBOL Mode

There is a new macro, disabled_in_cobol_mode$, that replaces the previous function named not_defined. Any user calls to not_defined will still execute the old function, which is still shared; however, all code in COBOL.EM and COBOL2.EM within EMACS*>EXTENSIONS>SOURCES has been changed to call the new disabled_in_cobol_mode$ macro.

New Command or Macro for Reestablishing Fundamental Mode Bindings

If the user has been working in some EMACS environment in which the terminal's function keys or ESCAPE or CONTROL-key bindings have been changed from their Fundamental Mode bindings, the fundamental command can be given to return to Fundamental Mode. This has two effects:

- It removes the bindings from all function keys on the terminal, if it has any. (The only exceptions to this rule are any terminal function keys which happen to send characters identical with Fundamental Mode keybindings. For example, the PT45 "SEND" key emits <CTRL>W. After the fundamental command, this key will be bound to the kill_region macro.)

- It reestablishes the Fundamental Mode keybindings for the ESCAPE and CONTROL-key commands.

The fundamental command does not destroy any macros currently defined. All it does is break any bindings between those macros and the ESCAPE and CONTROL-key keypaths.

A partial list of Fundamental Mode keybindings and associated Fundamental Mode macros is given in EMACS*>INFO>KEY_ASSIGNMENTS.EM. Note that if the user's library file has redefined any of the Fundamental Mode macros named in this file or bound by the other internal procedures used in the initialization of Fundamental Mode, the EMACS environment assigns the new definition to the Fundamental Mode keypath rather than to any other keypath being used before the fundamental command.

The new fundamental command may be given when the user is prompted by the extend_command command or with the syntax (fundamental) from the PL level.

See the related documentation on the new SUI and SUIX commands below.

CHANGES TO THE STANDARD USER INTERFACE

Change to PRIMOS_COMMAND Command

There is a new function, sui_primos_command, that has been bound to the <CTRL>X<CTRL>E keypath in SUIX. It has two advantages over its subset, the primos_command command:

- It displays an informative message:

  No prefix = Phantom; ! = internal, to file_output; !! = primos_internal_screen

  This at least partially explains the means by which users can execute the primos_internal_como or primos_internal_screen commands by prefixing the argument to this command with either one ! or two !! characters, respectively. See the earlier discussion of the primos_internal_screen command.

- Upon completion of whatever command was given, the new command executes the appropriate highlight function, restoring the reverse videoed screen command area.

Substitution of the FIND_FILE for the GET_FILE Function

The get_file function has been removed from the SUI. The Fundamental Mode find_file function has been put in its place.

With the newly available find_file function, SUI users are now able to inspect many files simultaneously without having to save one before looking at another. They are also able to use PRIMOS-level wild cards (@-signs) as arguments to the Find file: prompt and load several found files simultaneously.

SUIX users have more reason to take advantage of the <ESC>P, <ESC>N, <CTRL>Xb, and so on, commands.

The corresponding key of the PT200 SUI template is labeled FIND FILE instead of the old GET FILE. PT45 and PST100 users are encouraged to revise the GET FILE labels of their SUI templates.

The pros and cons of find_file vs. get_file are summarized in the new FIND FILE screen which replaces the GET FILE screen in the SUI HELP system.

The old get_file PEEL code has been left in the shared EMACS library so that any user wishing to keep on using it can reestablish the old keybinding. If the user wishes to return temporarily to the use of the old GET FILE function on the appropriate key, the set_get_file command is issued after pressing the COMMAND MODE key. Return with the set_find_file command. To permanently retain the binding of the unsupported get_file function to the GET FILE key on the terminal, add the following to the EMACS library file:

```
(set_get_file)
```

Addition of New Predefined PRIMOS Commands: LD, TLD, VLD

Previous releases of the SUI allowed the user to give the commands L or LISTF to the Command: prompt raised by pressing the COMMAND key. Either of these two inputs results in EMACS running the PRIMOS LISTF command as a primos_internal_screen. The LISTF is done at the current directory attach point.

Three new commands have been added to this release of SUI:

Name	Underlying PEEL
ld	(primos_internal_screen ld)

Thus this command simply runs the PRIMOS LD function at the current attach point and, when it is complete, the screen is restored to its state within EMACS.

tld (primos_internal_screen ld -long -srtd)

This command gives a time-ordered LD at the current attach point and then returns control to EMACS at its conclusion.

vld (primos_internal_screen ld -long -srtn)

This command gives a Verbose LD at the current attach point and then returns control to EMACS at its conclusion.

Of course, EMACS runs the appropriate command no matter if the user types in ld, LD, tLD, or VLd at the Command: prompt. EMACS is not case-sensitive when prompting for command names in the extend_command function.

All the above commands have one added innovation at this release. They add, to the above underlying PEEL, a prompt shown on the PRIMOS-level screen, at their conclusion. This instructs you to Press CONTROL-G to repaint the EMACS screen.

Changes to ATTACH and SPOOL Commands

The user feedback from the ATTACH command has been improved. If the attach is made successfully, the feedback to that effect is removed after sui_info_message_time$ milliseconds. If the user enters a directory which does not exist or can otherwise not be attached to, the screen is automatically cleared and the user gets a second chance to enter the directory name.

The SPOOL command has been similarly improved. If the file named in response to the Filename to spool: prompt does not exist, the user is given another chance. If the file does exist, the user is asked for additional SPOOL options to be given to PRIMOS. With this information, SUI proceeds to give the composite SPOOL command to PRIMOS as a primos_command that is executed as a phantom.

The SUI SPOOL command duplicates the actions of the S option in EMACS's EXPLORE display.

Miscellaneous Changes to Internals of the SUI

New Global Variable sui_info_message_time$: Many commands in the SUI display informational messages on the screen using the PL (info_message "text appearing on bottom line of screen") construct. A few of these messages, once displayed, remain there indefinitely, being removed only upon the next use of a function key. Other SUI commands display their info_message for a fixed length of time and then clear it away.

At this release of the SUI, the user can control the length of time that these info_messages remain on the screen. The controlling variable is named sui_info_message_time$ and it is set to a default value of 3000 milliseconds (3 seconds). Experienced users of SUI or SUIX may wish to shorten this value because it is a true time-out. During the time that this type of info_message is being displayed, user typeahead is often flushed depending on the function being used. To shorten the timeout period, for example to 1 second, add the following PL to your library file:

```
(setq sui_info_message_time$ 1000)
```

New Macro UNUSED_KEY_FEEDBACK$ and Its Relatives: Previous releases of the SUI had a macro that was misnamed do_nothing$. It performed the same type of thing that the new macro unused_key_feedback$ now does. Any user-written PL calling for the do_nothing$ macro will still find it available. All places within the SUI that used to call do_nothing$ now execute unused_key_feedback$.

There is a new macro, unused_mb_key_feedback$, that is bound to many function keys in mb_mode (that is, when they are pressed while the cursor is in the MiniBuffer). There is a new macro, eat_keystroke_no_feedback$, whose function is to accept an input keystroke with no error message or feedback of any type. Any user-written PL calling for the old null_function macro can still run the old function, which remains available, but all relevant SUI code now runs the new eat_keystroke_no_feedback$ macro.

Restriction on Many SUI Function Keys in the MiniBuffer: At this release, several SUI function keys on the PST100 and PT200 are bound to the newly-coined unused_mb_key_feedback$ macro for the purpose of restricting the types of actions that SUI or SUIX users can take while in the MiniBuffer. The original impetus for this was to eliminate the problem that occurred when the user pressed the HELP key when already in the HELP function. Previously, a user could invoke the SUI HELP while in the MiniBuffer, which is too small to hold the many display lines required.

The prohibition against use of certain functions in the MiniBuffer has been extended to the disabling of other keys that can potentially confuse the SUI or SUIX user if used there. Included in the disabled list are the CURSOR UP, CURSOR DOWN, INSERT FILE, SAVE FILE, WORD/CHAR, and several others. RIGHT and LEFT CURSOR, BEGIN/END LINE, DELETE and RUBOUT CHAR are not disabled so the user can edit MiniBuffer commands normally.

The user should consult the @BINDINGS.EM files in the EMACS*>EXTENSIONS>SUI>UNSHARED subdirectory for the full list of restricted function keys and change them in the private user library in order to use the MiniBuffer.

SUIX users who dislike the restrictions have another, simpler way around them. Use the Fundamental Mode keypaths, rather than the prohibited function keys, to navigate within the MiniBuffer.

Changes to Support Terminal Modes: It is possible, through mishap, that the PT200 terminal might get placed into its INSERT mode. In this case, the LED on the key etched INSERT (the SUI's OPEN LINE key) on the CF/N pad is lit up and the word INSERT appears on the PT200 terminal's System Line. If this happens, any depression of the OPEN LINE key is automatically bound to a new function to correct this wrong insert mode. It also does a screen clear to fix up any misdisplays that the wrong mode might have caused (EMACS normally operates in the terminal screen's default overlay mode).

It is also possible that the PT200 terminal might get placed into its NUM LOCK mode. In this case, the LED on the key etched NUM LOCK (the SUI's DEL FWD key) on the CF/N pad is lit up and the pad sends numbers instead of running EMACS functions. If this happens, any depression of this key is automatically bound to a new function to correct this wrong NUM LOCK mode. (If you are a Fundamental Mode EMACS user, or if you are writing your own EMACS interface, see the earlier discussion of the CF/N Pad's possible mode settings in Fundamental Mode.

It is also possible that the PST100 or PT200 terminals might receive an incorrect escape sequence from the host. This normally results in the terminal's bell sounding and the addition of INVALID CMD to the terminal's system line. At this release, there is a macro called remove_invalid_cmd$ that is bound to the RESET key on the PST100 and to the CLEAR key on the PT200. For the PT200 terminal only, the controlled augmentation of the PT200's CLEAR key sends <ESC>? that, in SUIX or Fundamental Modes only, is bound to the explain_key command. This saves a keystroke if all the user wants is to get a keypath explained. It is not necessary to type <CTRL>_C to get the same function.

The corresponding PST100 key, RESET, in its controlled augmentation, does a complete local screen clear. The RESET key, unaugmented, or <CTRL>L must be pressed to refresh the screen after that. This macro removes the error message from the terminal's system line and does a screen repaint so that any display that may have prompted the keypress is removed.

New Macro RESTRICT_TO_SUI$: All shared SUI macros that should not be run by non-SUI users have had a call to the new restrict_to_sui$ macro added to their code. Users without the proper internal variables set are not able to execute those SUI-specific macros if they bind them to keys or even if they invoke them by name. The reason for this is that the newly-restricted commands would fail less gracefully for the lack of atoms, which are available for SUI users but not for non-SUI users.

New Macro SUI_HIGHLIGHT: To facilitate the highlighting of the bottom three lines of the screen, a new PL-callable command, sui_highlight, has been added. This macro knows about the three physical and four logical terminals on which the SUI can be realized: PT45, PST100, PT200, and PT200W. It can be called without restriction on these four terminals. When called by a user without the proper underlying variable, keybinding$, set to either PT45, PST100, or PT200, this macro has no effect on the screen.

This new macro is called by the sui_refresh function. It is provided as a separate call for technical reasons that may be seen in various places in the EMACS*>EXTENSIONS libraries.

Fundamental Mode Commands Which are Replaced in SUIX Mode

The SUIX mode user will encounter some slight differences on certain Fundamental Mode keybindings. The SUIX user is accustomed to a certain type of service when the MARK key is pressed and probably would appreciate the same service when the <CTRL>@ key is pressed. In pure EMACS Fundamental Mode, this keypath is bound to the (setmark) function. However, for SUI, there exists the new function (mod_setmark). This latter function is assigned to the <CTRL>@ key in SUIX mode.

The TAB key, which always sends <CTRL>I to the host, and the BACK-TAB key (including whatever characters are emitted by this key on the three terminals supporting the SUI) are bound to TAB-related functions specific to the SUI and not those that are available to Fundamental Mode users. More of the differences are given in the following table.

Key Path		SUIX Mode Binding	Fundamental Mode Binding
<CTRL>@@		(mod_setmark)	(setmark)
<CTRL>X<CTRL>X		(sui_exchange_mark)	(exchange_mark)
<CTRL>X1		(one_file_mode)	(mod_one_window)
<CTRL>X2		(two_file_mode)	(mod_split_window)
<CTRL>XO	<UPPERcase O>	(mod_other_window)	(other_window)
<CTRL>Xo	<lowercase o>	(mod_other_window)	(other_window)
<CTRL>X{		(sui_horiz_left)	(horiz_left)
<CTRL>L		(sui_refresh)	(refresh)
<CTRL>X<CTRL>E		(sui_primos_command)	(primos_command)
<ESC>%		(query_replace_forward)	(query_replace)
<ESC>w	<lowercase w>	(sui_copy_region)	(copy_region)
<ESC>W			<UPPERcase W>(But not on the PT45 terminal, because of a conflict with the key etched FORMS, which sends <ESC>W and is bound to the SUI's global replace function.)
		(sui_copy_region)	(copy_region)

The bulk of the above changes are simply the replacement of a rather uninformative Fundamental Mode command with a SUI command that puts up a relevant info_message (mod_setmark, sui_exchange_mark, sui_horiz_left, sui_primos_command, sui_copy_region). The remainder of the above are real changes from the Fundamental Mode bindings done to improve consistency in the type of service expected (one_file_mode, two_file_mode, mod_other_window, sui_refresh, query_replace_forward). The last one, especially, is a substantial change from Fundamental Mode in that the fundamental mode query_replace is normally sensitive to the EMACS region, where the SUI's query_replace_forward command goes from point to the end of the buffer, regardless of the current region.

Horizontal Scrolling Has Been Added to the SUI

Background: One of the functional areas lacking in the SUI as originally released was horizontal scrolling bound to function keys. This has been now been corrected.

When EMACS SUI now comes up, the SCROLL LEFT and RIGHT functions are already bound to the function keys of all three SUI-supporting terminals.

PT45 and PST100 SUI users who don't add any customized keybindings will have to add some labels to their templates in the positions indicated by the keyboard maps shown in the HELP system and also in the newly-added HORIZONTAL SCROLLING item in the help menu. If users of these terminals don't do this, they may be surprised if they press the newly-active keys by mistake.

Terminal	SCROLL LEFT Key	SCROLL RIGHT Key
PT45	E-AUX	AUX-ON
PST100	PF4	PF6

PT200 SUI users' keys are already legended and all they need is documentation on the new keys' functions, which is also provided in the HORIZONTAL SCROLLING item in the HELP menu.

Users who have already made some additions to the PT45 or PST100 SUI or SUIX via -ulib files may have conflicts between their own keybindings and the keys which have been newly-assigned to SCROLL LEFT and RIGHT. There are two issues that must be addressed and both are trivially resolved. Since the new bindings are pre-shared, any user coming up through a private -ulib which binds something to the affected keys sees no change in those keybindings.

This is because the user's changes are done after the publicly-shared bindings are established in the product. Of course, they lose the newly-available ability to SCROLL LEFT and RIGHT on those keys but at least their custom-built interfaces are not disrupted.

An existing -ulib user, with conflicts on the assigned scrolling keys, gets horizontal scrolling by:

- Moving their own functions off the newly-assigned SCROLL LEFT and RIGHT keys and allowing the shared bindings to take effect
- Binding these sui_horiz_left and horiz_right functions to unused keys on their terminals

If the user wants to bind the horiz_right and sui_horiz_left functions to keys of their own choosing, see below for descriptions of files containing lists of the preshared EMACS variable names to use in doing these bindings. (The sui_horiz_left function is to be assigned to the SCROLL RIGHT key. The name conflict is because the FUNCTION NAME is talking about moving the WINDOW rather than moving the TEXT, which is the naming standard used in the SUI.)

Functional Specification of SCROLL LEFT and SCROLL RIGHT Keys: The basic specification is given in the following excerpt from one of the HELP screens.

SCROLL LEFT: Means "Push the TEXT to the LEFT so I can see more toward the right end of the lines." Unless you say otherwise, the horizontal offset changes by 40 columns. If you prefix the SCROLL LEFT key with ESC followed by a number — e.g., ESC 23 SCROLL LEFT — you will get put on "hcol 24."

SCROLL RIGHT: Means the opposite. This key also takes the ESC-number prefix.

If hcol is very large and you want to scroll RIGHT "all the way," press ESC 1 and then the SCROLL RIGHT key, and hcol will be set back to 1.

When hcol is greater than 122 at the conclusion of a SCROLL RIGHT command, a helpful hint appears for a period of time equal to sui_info_message_time$ milliseconds telling the user how to get back to zero offset without many additional SCROLL RIGHT keystrokes.

Change in SUIX RPG Mode on PT45 Terminals: Due to the addition of the SCROLL LEFT and RIGHT keys to the PT45 SUI, there has been a displacement of the RPG Mode function key which used to be bound to the rpg_unplate$ function. This used to be bound to the E-AUX key; it has now been moved to the PAGE key.

Changes to the SUI HELP Screens

HELP screens and function key maps appropriate for the PT200 in its 80x24 display mode have been added to the HELP system. There are

changes in several of the "Explanation of Key Functions" screens, including substitution of a FIND FILE screen for the old GET FILE screen, including some of the advantages of FIND FILE over the old GET FILE function.

SUI Support in Both Display Modes for the PT200 Terminal

SUI/SUIX is designed for display widths of only 80 or 132 and nothing in between. SUI/SUIX is designed for display heights of only 24 or 27 and nothing in between.

Normal Operation of SUI/SUIX on PT200 Terminal

SUI/SUIX operates (virtually) identically in both 80- and 132-column display modes. The only major difference is that what is normally seen in 80-column display mode is seen compressed in 132-column mode.

Keyboard Layout for PT200 SUI: This is available, for the most part, by inspection of the PT200 SUI keyboard template. Read the SUI template legends, not the keytop etchings.

CHANGES IN THE EMACS* DIRECTORY

The major user-visible changes in the structure and content of the top-level EMACS* directory are presented here.

Contents of EMACS*

TEACH-EMACS Available for SUIX Mode: The only files in this top-level directory are the TEACH-EMACS@ files, computer-assisted instruction of EMACS's Fundamental and SUIX Modes. At this release, the older @FUND@ files have been renamed, reorganized, and rewritten. Additionally, there is a whole new group of several @SUIX@ files, similar in content to the @FUND@ files but designed for training users on the SUIX.

Subdirectories for Special Use: Below this directory are the subdirectories EXTENSIONS, INFO, and TERMINAL_HANDLERS1. The latter directory is for the use of the local EMACS maintainer and user community as a place where customized terminal drivers are kept.

If the LIB subdirectory is also present, it is for the use of the local EMACS maintainer and community.

> **Caution**
>
> The contents of EMACS*>LIB may be erased as new versions of EMACS are installed, although the newer versions of the EMACS installation files are designed to prevent this.

Contents of EMACS*>INFO

All files in this subdirectory whose names terminate in _INFO are screen files used by the EMACS SUI HELP system. The only exception to this rule is the describe_emacs file. There have been several PT200-related additions. The obsolete sample_library_file.em file has been deleted at this release.

Contents of EMACS*>EXTENSIONS

The .EFASL files found in this subdirectory comprise the shared code that is used in EMACS Fundamental Mode and in many language-specific Modes. All .EFASL files in EMACS*>EXTENSIONS may be considered resident in shared memory.

Contents of EMACS*>EXTENSIONS>SOURCES

All the source files that produce the .EFASL files above are in this subdirectory. This code can be an extremely valuable tutor on the fine points of the Prime EMACS Extension Language, PEEL, or PL. Many users have reported that EMACS's PL was obscure until they saw the library sources.

The EMACS*>EXTENSIONS>SOURCES subdirectory now contains the files: PST100_FUNCTION_KEYS.EM, PT200_FUNCTION_KEYS.EM, PT45_FUNCTION_KEYS.EM. These are the PEEL source files giving the symbolic names of the function keys for the three Prime terminals supporting SUI/SUIX.

These files make it considerably easier for users to create their own customized EMACS interfaces because their purpose is to save the user the work of finding out exactly what each terminal key sends to the host. In the shipped version of EMACS, the actual keybindings in SUI and Fundamental Mode are not done with these symbolic files. They are established by an automatic setup procedure. However, the intentions of the files have in fact been executed as part of EMACS's initialization sequence, and therefore the user can establish custom bindings to those variables in the way described in the beginnings of each of the above noted files.

SOFTWARE PROBLEMS FIXED

Executing a PRIMOS Command no longer requires attribute modification rights in the current directory. All intermediate phantom or comoutput files used during this operation are put into the user's initial login directory (SPAR 3000531).

The EMACS allocator aborts the command but stays in EMACS if given a request for more than 32K words (SPAR 3002340).

The redisplayer is now fixed to correctly display the first screen position if the window is horizontally scrolled and the character in the first screen position is the last character on the line in the window (SPAR 3002380).

User error (SPAR 3002713).

RPG mode, as of Rev. 19.3, uses VRPG as the default and old RPG as a request option. COBOL mode defaults to COBOL (old COBOL). An extension of .CBL defaults to new COBOL (CBL) (SPAR 3003545).

The forced logout condition causes EMACS to save all modified buffers into the user's initial login directory (no change from previous versions) but the mechanism has been changed to allow the user name and the initial login directory to be different. A new message is output giving the pathname of the file to which each buffer was saved. The CPL file output to copy the buffers to their original places in the file system hierarchy has been fixed to work correctly (SPAR 3004514). A bug of SpeedType has been fixed. If a user added the same abbreviation twice without deleting it, an Access Violation occured. This has been fixed. This was also SPAR 3007420 (SPAR 3004517).

The closing of LISP expressions by the reader, which is what occurs in the minibuffer and when a buffer is compiled, is considered a feature and not a bug (SPAR 3004519).

The new COMPILE mode uses an improved error scanner which can interpret the output from PL/1, PL/1G, SPL, CBL, VRPG, Pascal, C, RPG, PMA, COBOL, F77, and FIN (SPAR 3004521).

In FORTRAN mode, continuation of comment lines longer than 72 characters was incorrect. This has been fixed (SPAR 3004556).

The new COMPILE mode allows users to add or override established compile options by including new options on the compile command line (SPAR 3004559).

EMACS now clears protect mode on a NL8009 terminal during initialization (SPAR 3004600).

The PL command now gives the error 'Cannot PL, buffer too big' if the buffer is too big (more than 32000 characters) (SPAR 3005943).

GET_FILE incorrectly handles rereading a file with the same name as the current file in another UFD. Read the information on GET_FILE in this document; it is considered obsolete (SPAR 3006014).

The &repeat option works as documented (SPAR 3006091).

The problems with search have been fixed. The user should be warned that code written to use the old search functions may now stop working (SPAR 3006092).

The complete list of changes follows:

The primitives:

verify_fd	(formerly search_not_charset_forward)
verify_bk	(formerly search_not_charset_backward)
verify_fd_in_line	(formerly search_for_first_not_charset_line)
verify_bk_in_line	(formerly search_back_first_not_charset_line)

previously returned false only if all the characters to end of the buffer (line for "..._in_line" functions) were members of the character set specified. In this case, the cursor was moved to the end of buffer (line). All other cases returned the value true, regardless of whether the cursor was moved or not.

The documentation states that these primitives "...return a boolean that indicates whether or not the appropriate character [in the set] was found." All the verify_ primitives now return true if a character in the specified set was found and false otherwise. Note that this leads to the correlary that true equals cursor moved; false equals cursor stationary.

This returned value is consistent with the set of "search_..." routines, which also return true if an appropriate character in the specified set was found and false otherwise.

The fixes here also affect the return value of the following primitives, which call upon the "verify_..." routines and return their returned value:

skip_over_white
skip_back_over_white

The primitives:

verify_fd	(formerly search_not_charset_forward)
verify_bk	(formerly search_not_charset_backward)
verify_fd_in_line	(formerly search_for_first_not_charset_line)
verify_bk_in_line	(formerly search_back_first_not_charset_line)

previously ignored a NewLine "^n" character within the buffer, implicitly including it in the character set to be verified. They no longer include the NewLine character implicitly and will perform correctly when it is included explicitly.

The fixes here affect the following internal primitives since they directly use the "verify_..." routines:

skip_over_white
skip_back_over_white

Previously, these routines implicitly included the NewLine as a white-space character to be skipped over even though the default value for the whitespace global did not include it. This was contrary to the Emacs documentation. Now they work as documented.

The primitives:

search_fd	(formerly search_charset_forward)
search_bk	(formerly search_charset_backward)
search_fd_in_line	(formerly search_for_first_charset_line)
search_bk_in_line	(formerly search_back_first_charset_line)

previously ignored a NewLine character within the buffer, implicitly excluding it in the character set to be searched for, even if that character set explicitly included a NewLine. They no longer exhibit this implicit behavior and will honor an explicit NewLine correctly.

The fixes here affect the following internal primitives since they directly use the "search_..." routines:

skip_to_white
skip_back_to_white

Only if the whitespace (global) string contained a NewLine is their behavior now different. Since the whitespace string defaults not to include a NewLine, their behavior should not normally be affected.

The primitives:

forward_search
reverse_search

previously would behave unpredictably and inconsistently when a NewLine character was included as the first character of a search string. They now work properly.

The following primitives are directly affected:

^s_forward_search_command
forward_search_command
reverse_search_command
query_replace
replace

The primitives:

forward_search
reverse_search

previously could not locate the last (first with reverse_search) text in a buffer when the NewLine character was included as the first (last) character in the search string and remaining characters spanned the last (first) character in the buffer. Both primitives now behave correctly at buffer boundaries.

The following primitives are directly affected:

^s_forward_search_command
forward_search_command
reverse_search_command
query_replace
replace

Get_file is obsolete. Find_file, on the same function key, correctly loads a file in two file mode after a pathname error on the first try. Read the information on GET_FILE in this documentation (SPAR 3006770, 3008712, 3008806).

Fixed a bug which allowed "gutters" in fill mode (SPAR 3009063).

Fixed call to RDLN$P in READF. User no longer gets remainder of old read buffer after aborting find_file (or read_file, insert_file) (SPAR 3006090).

Fixed dumpfile to correctly determine the pathname of the file to be dumped (SPAR 3006258).

Fixed so successive yank_replaces separated by blanks do not behave the same as those separated by other characters (SPAR 3007488).

OUTSTANDING PROBLEMS

Incomplete documentation in PRIMER and REFERENCE GUIDE (SPAR 3001737).

On a PT45 in VIEW mode, tabbing past the end of the line locks the keyboard, rings the bell, and displays an error message (SPAR 3003146).

QUIT during initialization leaves the terminal in half duplex mode (SPAR 3003437).

Documentation bug in the extensions guide incorrectly states that the PEEL function make_array produces an array with bounds (0..N) when in fact the bounds are (0..N-1),that is, an array of N elements (SPAR 3004515).

Pericom78 terminal driver switches the numeric keypad to function keys (SPAR 3004605).

Problem in COMPILE if level 'd' errors occur (SPAR 3006256).

When a library .efasl file is not found while installing EMACS, it causes the operator console process to get a fatal error and be reinitialized (SPAR 3006785).

If the command line argument -HEIGHT is given an argument greater then the physical height (or diplay height) of the screen, the EMACS display may act strangely. A similar situation holds for the -WIDTH argument (SPAR 3009068).

PERMANENT RESTRICTIONS

PL can only compile buffers with less than 32000 characters in them. This is a permanent restriction (SPAR 3005943).

Rebinding of the function forward_place_holder to a printing key incorrectly removes the place holder (SPAR 3006257).

Forward_kill_sentence and forward_kill_clause cannot collect successive kills even if the cursor is not moved between them. These are library functions and the cursor does, in fact, move, processing them even though that motion is not displayed (SPAR 3006599).

INSTALLATION AND BUILD PROCEDURES

Standard.

Note

EMACS uses segments in the 2000 range for its libraries. At Rev. 19.4, EMACS allocates storage in the dynamic segments.

FIXRAT

NEW FEATURES AND CHANGES

The readablity of the user HELP frame is improved.

FIX_DISK

NEW FEATURES AND CHANGES

New error code has been added for use by DAM_WALK to report to parent that a zero record was added (a warning code only).

SOFTWARE PROBLEMS FIXED

Added checking for correct value of dam index level in non-dam files (SPAR 3007446).

The record-time product is correctly calculated (SPAR 3009239).

A file is now always marked as truncated by FIX_DISK if zero records are added to the file (SPAR 3005641).

ICS1

NEW FEATURES AND CHANGES

The ICS1 now performs asynchronous reverse flow control thus controlling flow of data from terminals.

SOFTWARE PROBLEMS FIXED

Outbound flow control support was added to the ICS1.

Code was added to mask out unused bits when less than 8 bits are received from the terminal.

The state of the RTS value is changed in to RTS on.

Code was added to count the number of receiver overruns on each line.

Code was added to reschedule the queueing of an entry into an event queue.

All fixes included in Rev. 19.3 fix releases up to 19.3.2 have been included.

PERMANENT RESTRICTIONS

Flow control cannot be used with terminals that do not support the standard XOFF and XON characters. If it is used with any of those terminals, data from those terminals might be lost during a flow control situation. Up to 16 characters per terminal are buffered when lines get flow controlled. If all 8 lines become flow controlled, the last line is only able to buffer 15 characters.

ICS2

NEW FEATURES AND CHANGES

Asynchronous reverse flow control is added.

SDLC and combined SDLC/asynchronous protocols are now supported.

The following multiple download files for different protocol combinations are added:

ICS2_01.DL (async)
ICS2_02.DL (SDLC)
ICS2_05.DL (async + SDLC).

The buffer server process returns transmit buffers on the high priority queue to prevent deadlock.

There is console monitor support for PT200 terminal.

The following new console monitor commands are supported:

- SIO/SCC read register (RR)
- Display synchronous lines (SY)
- Display physical line (PL)
- Display free pool(s) (FP)

SOFTAWRE PROBLEMS FIXED

The following software problems are remedied:

- Asynchronous initialization on the second through the fourth ICS2 controller
- Bad status reporting to PRIMOS (SPAR 3004717, 3005114, 3005117)
- Character handling with or without parity enabled
- RTS handling holds RTS high instead of low
- Problems with asynchronous loopback and echoback modes
- Problem with SDLMTR line statistics

- Master logical connection for reverse notifies when queue in Primos goes from full to not full
- The queuing routines for one specific case
- A potential one word status buffer write by the IBC
- Additional error handling for invalid IBC program counter implemented

LD

NEW FEATURES AND CHANGES

Output formats have been changed to take up less space. No entry ever requires more than two lines. The -DTM, -DTB, and -SIZE options use only one line when invoked individually. Some combinations of options only require one line as well.

"Date time saved" is changed to "Date time backedup" (-SORT_DTB, -SORTB). This feature is reserved for future use. A "Date time backedup" column containing '** dtb not set **' is currently displayed.

The size and quota fields are expanded to seven digits to support future expansion of maximum file size.

A new option, -NO_COLUMN_HEADERS (-NCH), is added to allow suppression of column headers in detailed output modes.

SOFTWARE PROBLEMS FIXED

Fixed bug which would output 'No entries selected' randomly even when the UFD contained data. Initialized TABLE_FULL variable (SPAR 3009226).

The selection of an output format string is now a simple array operation not a massive SELECT statement. A few flags were removed and comments were properly indented.

LOAD

SOFTWARE PROBLEMS FIXED

There is a new method for determining if the map file is already open.

PERMANENT RESTRICTIONS

LOAD does not always return a severity code for all problems.

LOGPRT

NEW FEATURES AND CHANGES

The following changes were made:

- An error is entered in the network event-log file when a bad HDX restart packet is received by the DTE.
- Support is added for 9955 and 9750.
- The ECCULO event type is supported.
- Messages for ICS2 are updated.

In addition, LOGPRT is changed to recognize recoverable machine checks. It is modified so that the routine that decodes the contents of DSWSTAT for P9950's checks if bit 26 of DSWSTATL is set. If the bit is set, these utilites have been modified to indicate that the machine check was recovered. The decode routine for the P9950 will also be used by future processors.

MAGSAV MAGRST

NEW FEATURES AND CHANGES

In-use EPF's are restorable with the in-use copy having its suffix changed to .RPn (n ranges from 0 through 9).

The existing -QUERY command line option tells the user which replace file has been created. If all ten in-use EPFs with the filename suffix .RPn exist, one will be overwritten if the user does not use the -QUERY option. With -QUERY, the user is prompted for which old replace file is to be deleted. Users should delete the .RPn files as they accumulate.

The list of index filetypes has been extended to cover the filetypes DAM-RBF, SAM-RBF, SEGSAM-RBF and SEGDAM-RBF.

The warning message for no write-ring on the tape has been altered from 'Tape is file protected' to 'Tape is write protected'.

Spelling corrections were made to usage text.

The message 'We are ignoring it', to indicate that a ROAM file from tape is not being written to disk, is replaced by the message 'File not restored'.

SOFTWARE PROBLEMS FIXED

The test for the BADSPT file has been improved so that MFD files which begin with the characters BADSPT, such as BADSPT.FIN, are restorable and files named BADSPT are restorable at UFD level or lower. The BADSPT file on the MFD is still not restorable (SPAR 3002715).

If the -TTY option is used and the tape unit is off line or write protected, the program loops back to the 'Tape unit:' prompt for command input files and CPL files (SPAR 3002990).

Unrecoverable write errors during the save of a SEGDIR formerly restarted the save from the beginning of the SEGDIR. When MAGRST encountered a situation of being in one SEGDIR and finding another, it attempted to attach down a level and failed.

On an unrecoverable error during a save of a SEGDIR, the save is restarted from the subfile before the one in which the error occurred. On restoring a SEGDIR which spans reels, re-initialization is performed, preventing any bad attaches (SPAR 3003287).

A space is not taken to be equivalent to <RETURN> for the 'Enter logical tape number:' and 'Revno:' prompts (SPAR 3003890).

Partial restores of ROAM files using a full treename formerly caused fatal errors because of an attempt to attach to the ROAM file itself. If the user does a partial restore of just a ROAM file, the attaching routine MGATCH is not used, preventing the bad attach (SPAR 3004875).

Files of the types SAM-RBF and DAM-RBF can be created if the files did not previously exist and if the logical file bit is set as an attribute.

If an RBF file of the same type exists, the file is not restored and an "Improper access of restricted file" message is given. If the file exists, but only one of the disk and tape files has its RBF bit set, a mismatch error message is displayed and the file is not restored. This enables the ROAM system files to be saved and restored in the event of disk reformatting (SPAR 3005056).

Duplicate copies of the same file, identical in all respects, formerly produced an error message of "Unit not open" on alternate copies. This was caused by the name checking algorithm believing the second copy to be a sub-entry of the first. The routine then attempted a SEGDIR search for that sub-entry, the SEGDIR being the not open unit.

The algorithm has been modified and all identical copies of a file are now restorable (SPAR 3006149).

The variable holding the severity to be returned is initialized to zero on entry to MAGSAV. This prevents the program from being run once with a bad tape unit being supplied and finishing with an ER! prompt followed by the same set of commands returning to an OK, prompt.

If such a sequence of commands were in a CPL file, which caught errors and summed them up, the first run would be registered as an error but subsequent invocations would not register as errors (SPAR 3006239).

Partial restores of an existing tree structure, containing ROAM files on lower levels, failed to work due to incorrect building of the full relative pathname and due to the ROAM routine DLSD$U always attaching back to the home directory.

The required portion of the restored file's pathname is now built. This allows the DLSD$U routine to delete the existing ROAM file from the disk. When this routine returns, MAGRST attaches back to the correct location for restoring the master file (SPAR 3006574).

If ROAM is not shared and the user attempts to restore a ROAM file, the correct filename is given and the ROAM library routines are not called at all. This prevents a fatal LINKAGE_FAULT$ condition from occurring (SPAR 3006842).

If, for some reason, a ROAM file is not restored, a routine is used to scan through the tape until either a treename or an end of tape is found. At this point, the routine returns to the main program and processes data as before. This routine was written to ignore all MASTER records and all SLAVE records.

However, if there were consecutive ROAM files on tape and the first was ignored, the second file would also be totally ignored. The routine now ignores only MASTER records until the end of the current ROAM file on tape (SPAR 3007096).

Nested SEGDIRS are not currently supported. Introduction of this feature is anticipated at a later date (SPAR 3003639).

PHYSAV PHYRST

NEW FEATURES AND CHANGES

The routine AYENAY, for obtaining yes/no responses, has been modified to accept answers in upper or lower case or a mixture of both. Most of the messages and prompts are now given in lower case.

Other instances of character string interpretation; for example, the RESTORE/VERIFY option of PHYRST or the QUIT option on the tape unit number prompt, are also no longer case sensitive.

The PHYSAV utility now uses the non-pure FORTRAN library. It formerly used the default library, LI, which is now an EPF; the EPF libraries reside on the command disk. This enables the command device to be saved via the process of starting PHYSAV and naming the command disk using its pdev. This causes the error message "Disk not assigned" to be displayed and the user is returned to PRIMOS. The disk should then be shut down, added to the assignable disks table, and assigned. PHYSAV can then be restarted using the START command and the save progresses normally. At the end of the save, the disk should be added back to the system using the ADDISK command.

SOFTWARE PROBLEMS FIXED

A space is no longer taken to be equivalent to <RETURN> for the 'Restore partition nnnnnn as partition:', 'Logical tape:', and 'Phys.Dev.No:' prompts (SPAR 3006575).

When the -TTY option is used, PHYRST takes its magtape unit number from the terminal rather than from the current command stream (SPAR 3007945).

PRINT_NETLOG

NEW FEATURES AND CHANGES

An error is put into the network event-log file when a bad HDX restart packet is received by the DTE.

PRINT_SYSLOG

NEW FEATURES AND CHANGES

PRINT_SYSLOG has been changed to recognize recoverable machine checks. It has been modified so that the routine that decodes the contents of DSWSTAT for Prime 9950's checks to see that bit 26 of DSWSTATL is set. If the bit is set, these utilites indicate that the machine check was recovered. The decode routine for the Prime 9950 will also be used by future processors.

SPOOL

NEW FEATURES AND CHANGES

A number of enhancements have been made to the spooler at Rev. 19.4 to overcome problems experienced in the past. In addition, a number of bug fixes have been incorporated. The most fundamental change has been to incorporate the OAS spooler phantom into the standard spooler phantom. This has resulted in a few more printer types in the PROP environment and an additional file. L.TYPE, in SPOOLQ.

For those users who wish to operate with passworded SPOOLQs, the password needs to be changed in only one place and the spooler reloaded. There is a CPL program, CHG_PWD.CPL, to perform this function. It prompts for the new password, edits the single source program, and reloads the spooler. The spooler then has to be reinstalled and SPOOL$ has to be reshared.

New SPOOL Options

Eight new spool options have been added to the SPOOL command and to the SPOOL$ subroutine and some improvements to existing options have been made.

-DISK diskname or ldev: This feature allows the user to specify which partition to spool a file on, allowing multiple SPOOLQs on a system and thus removing the limitation of 200 entries in a system's spool queue. This option can be used to spool files on both local disks and remotely added disks, providing the disk contains a SPOOLQ directory. Using ldev is the faster of the two methods.

-MODIFY PRT_filenumber or -MOD filenumber: This option allows the user to modify an existing entry in the SPOOLQ. All normal spool options are permitted to be used with -MODIFY. The format of the command is:

```
SPOOL -MODIFY PRT002 [options]
```

or

```
SPOOL -MOD 2 [options]
```

The options are the normal SPOOL command options with the usual restrictions. Please note that only the options that are specified with -MODIFY are changed in the print request.

-NODEFER or -NOD: Used in conjunction with -MODIFY, -NODEFER removes the -DEFER attribute from a job.

-NOEJECT or -NOJ: If specified in the command line, -NOEJECT disables the form feed that normally occurs after the file has completed printing. Please note that a form feed is never performed before the banner page and thus this option should only be used in circumstances where the banner page is inhibited with the -NOHEADER option.

-NOTIFY or -NFY: Notifies the user when the user's file has completed printing. Notification is done by the spooler phantom by first attempting to send the user an immediate message. If the message is not received, it will be resent as a deferred message. Since the notification is done by the phantom, the user will be notified only if a Rev. 19.4 or later spooler phantom prints the job.

-RUSH/-NORUSH: The spooler phantom, while scanning a SPOOLQ directory, first checks to see if there are any priority files in this particular SPOOLQ. If there is one, that file will be printed next regardless of size restrictions or its location in this SPOOLQ. Form type and printer destination are adhered to. Where more than one priority file exists in a SPOOLQ, the priority files will be printed in the order spooled. This option can only be used in conjunction with the -MODIFY option. Since the rush is done by the phantom, the file will be rushed only if a Rev. 19.4 or later spooler phantom prints the job.

For example:

```
SPOOL file1 -FORM WHITE   (assume it becomes PRT001)
SPOOL -MOD 1 -RUSH        (to set priority)
SPOOL -MOD 1 -NORUSH      (to remove priority)
```

Only user SYSTEM and those users listed in the files L.USER and L.environment are permitted to use -RUSH and -NORUSH.

-TRUNCATE or -TRU: Prevents a file containing long lines from printing on the platen. Truncation is performed by sending only the first n characters of each line to the printer. n is defined by the PROP WIDTH command. Since the truncation is done by the phantom, the file will be truncated only if a Rev. 19.4 or later spooler phantom prints the job.

SPOOL -LIST AT <printer_name>: Lists those files which are spooled to a specific printer. This can be used in conjunction with the -LIST * command:

```
SPOOL -LIST * AT <printer name>
```

Cancelling a Print Request: If a SPOOL -CANCEL command is issued and the file is printing on the local system, a DROP command will be issued to the printer concerned. Please note that files that are printing on a remote system cannot be dropped; in that case, the following error message is displayed:

File is printing on a remote system - unable to drop it.

Changes to SPOOL$

The size of the INFO array has been extended to allow the inclusion of the extra SPOOL options. All the additional spool options are available under direct calls to SPOOL$. The details on the calling sequence for SPOOL$ are included at the end of this document. Please note that Rev. 19.4 SPOOL$ is totally compatible with all previous versions of SPOOL$.

On a key of 2 (open file), SPOOL$ finds an available unit number to open the file on if the value supplied in INFO(2) is zero.

An additional key of 4 has been included that closes the unit opened by a previous invocation of SPOOL$ with a key of 2. Users formerly had to close the unit manually by calling SRCH$$ after writing to the open unit. This second invocation of SPOOL$ also notifies the spooler phantom semaphore.

Changes to SPPHN

If the QSCAN parameter in the PROP environment is set to 1, all SPOOLQs are now accessed equally. This means that, having printed an item from the first SPOOLQ on the system, SPPHN scans the next SPOOLQ for a file eligible to be printed on that printer. This significantly speeds up the printing of those files which reside on SPOOLQs other than the first queue and this option should be selected at sites where more than one SPOOLQ exists to be scanned. Please note that this can cause delays of a few seconds between printing files if secondary SPOOLQs are on remote systems since the SPPHN phantom has to do a remote attach between the printing of the files. This can be avoided by the judicious use of the UPPER and LOWER values in the PROP environment.

Please note that for those sites where the old method of searching is sufficient, QSCAN should be set to 0 (the default).

An ETX/ACK protocol has been incorporated for OAS printers.

If COMO ON is specified, the command output file is protected for n readers and one writer (as long as the phantom has protect rights). This allows users to list the COMO file while the printer is printing.

If the phantom is unable to assign the AMLC line at startup, an error message is output and the phantom is logged out.

Trailing blanks are stripped from output lines. This prevents needless output of characters and removes the vertical bars that are the representation of a space on Printronix printers using plot packages.

Lines up to 400 characters in length are now printed.

> WARNING
>
> For printers on MPC boards, the PRIMOS subroutine T$LMPC truncates lines at 140 characters.

On completion of printing a file, a call is made to a routine ACCTNG. A 200 word buffer containing information on the just completed print file is passed to this routine. The actual structure of the buffer is defined at the end of this document, but among the information passed is the number of pages printed, the number of lines printed, and the start and stop time for printing. It is intended that users write their own routine to extract the information pertinent to them and store it in any format or file they require.

Adding Electronic Vertical Format Units (EVFU) to a printer environment is now performed correctly.

Characters without parity set are now converted to uppercase. This required a change to UPCAS$.PMA.

A bug was fixed that caused files spooled in NOFMT mode to start on the second line of a page. Files spooled with -NOF,-NOH,-NOE are now printed without a blank line between them.

Changes for OAS

An OAS printer is recognized by a PROP TYPE specification ranging from 10 through 15. Currently there are three OAS types:

10 - OAS, Manual
11 - OAS, Cut sheet feeder
12 - OAS, Tractor Feed (Continuous)

In MANUAL mode, all form feeds (FF) are replaced with a null and the STOP code sequence for the printer is output. The printer pauses and the operator is then free to change or insert new paper and continue the print operation in the normal manner as defined by OAS.

In the case of a printer defined as a KSR (see file L.TYPE), the signal to continue is taken as a character input from the keyboard of the printer.

For a cut sheet feeder, the very first FF in the file is not output unless a header has already been output. This prevents a blank sheet of paper from being thrown out each time a file starts printing. All other FFs are acted on normally. A tractor feed (continuous) printer is the same as a standard line printer.

OAS uses the ETX/ACK protocol rather than XON/XOFF. A new driver value of 6 indicates that the ETX/ACK driver is to be used. This is set by SPPHN after checking the printer type.

The normal spooler outputs these messages (in large letters) to the printer. For OAS printers, they will be output only if the printer is a type 12 (continuous).

Changes to PROP

Compressing the Queue: A new facility exists that allows printer operators to tidy up the SPOOLQ on any disk. Printer operators are defined as those users listed in the file L.USER. The new facility is designed to perform several functions:

- Remove any entries from the queue that do not have a corresponding PRTnnn file
- Delete any PRTnnn file that does not have a corresponding entry in Q.CTRL
- Reorganize the queue to maximize the use of available slots
- Make the secondary entry area for Rev. 18 entries nulls and spaces; that is, make it as a Rev. 19 entry

The file Q.CTRL is held open during the entire running of this program. Thus no user or spooler phantom is able to access the queue while this program runs. This facility provides welcome relief to the age old problem of 'No room' when in fact there are far less than 200 entries in the queue.

To invoke this facility, type:

```
PROP -COMPRESS [ diskname ]
               [ ldev     ]
```

If no diskname or ldev is supplied, PROP defaults to the first SPOOLQ.

Naming the Phantom: PROP has been modified to start the phantom via a call to BATCH$. This allows the phantom to be named explicitly; it is given the name of the printer environment. If the command is not issued from the supervisor terminal and therefore fails due to 'no access rights', the phantom is started directly. (See the WARNING on the next page.)

> WARNING
>
> This may affect access control lists if ACLs are used on the SPOOLQ.

PROP now makes provision for a passworded SPOOLQ and performs an 'ATTACH SPOOLQ xxxxxx' in the command file to start the phantom.

Printer Control: To allow the System Administrator to exercise more control over which users are permitted to control the printers, two new files can be created in the SPOOLQ UFD. L.USER is a general file that contains usernames (upper case) of those users who are permitted to START, STOP, MODIFY and CREATE the printer environments. An L.environment file can also be used to restrict access even further for that specific printer. The L.environment file is checked first and, if it does not exist, the L.USER file is checked. If neither file exists, only the user SYSTEM is permitted to START, STOP, MODIFY and CREATE the printer environments.

No change has been made to the length of the printer environment files; thus printer environments are compatible with all Rev. 19 environments.

To overcome the problem of PROP commands timing out when using slow printers, a change was made in the spooler phantom. The phantom now checks for PROP commands every 16 lines for normal printers and every 8 lines for OAS printers.

Printer Types: Some additional printer types have been added. Permitted values of TYPE are now 0, 1, 10, 11 or 12. These types are:

0	Printronix (300 lpm printer/plotter)
1	Dataproducts (band printer)
10	OAS, Manual
11	OAS, Cut sheet feeder
12	OAS, Tractor Feed (Continuous)

PROP Options: PROP -ABORT, PROP -DROP, or PROP -RESTART clear pending hangs and cause an implicit -CONTINUE. They also cause a pending -STOP or -MODIFY to take place immediately.

PROP -MODIFY NOW causes the file currently printing to be aborted. The modification then takes place and the next applicable file starts to print (it could be the same one depending on what has been modified).

PROP Subcommands: The default values of WIDTH, LINES, and LENGTH have been changed. The default values are now 132, 66, and 60, respectively. These values are commonly used with the wide, 14 by 11 inch, paper. A new entry in the PROP environment permits the user to suppress the printing of the warning messages on the printer relating to stopping, aborting, dropping, and so forth. The new entry is WARNING and takes a value of ON or OFF.

An additional entry has been made to the PROP environment to determine how the spooler phantom should access systems where there are more than one SPOOLQ. The entry is QSCAN and takes the value 0 or 1.

If QSCAN is set to 0, the SPOOLQs will be scanned as follows. Files spooled to the local, or first, SPOOLQ are printed first. When this queue is empty, the second SPOOLQ is accessed. If a file exists in this queue, it is printed. The first SPOOLQ is rechecked and any files in it are printed before the second SPOOLQ is rechecked. When both the first and the second SPOOLQs are empty, the third SPOOLQ is checked, and so on. This method is not recommended for systems with multiple SPOOLQs.

If QSCAN is set to 1, all SPOOLQs will be accessed equally. This means that having printed an item from the first SPOOLQ on the system, SPPHN will then scan the next SPOOLQ for a file eligible to be printed on that printer. This significantly speeds up the printing of those files which reside on SPOOLQs other than the first queue and this option should be selected on sites where more than one SPOOLQ exists.

SPOOL$

Below is the new definition of SPOOL$ that includes all the extra features incorporated in the Rev. 19.4 version of the spooler.

A user program can insert a file into the spool directory by calling the SPOOL$ subroutine:

```
CALL SPOOL$ (key, name, namlen, info, buffer, buflen, code)
```

key	User option
1	Copy named file into queue.
2	Open file on unit info(2) for writing. If info(2)=0 then SPOOL$ will find an available unit and return the value in info(2).

3	Modify the attributes of a file already spooled.
4	Close file on unit info(2) in queue and notify semaphore.

name File to be copied (if key = 1), or name to appear on header page (if key = 2).

namlen Length of name, in characters (1-32)

info Information array, 40 elements, as follows:

1	Reserved after Rev. 17.
2	Open print file on this unit (key=2) If unit number is zero then SPOOL$ will return the unit number here.
3	Print option word. (See below.)
4-6	Form type (6 ASCII characters). This must be entered and blank filled (equivalent to -FORM on PRIMOS command line.)
7	Plot raster scan size (plot only). This represents number of words/raster scan.
8-10	Spool filename (returned).
11	Deferred print time (valid only if defer bit specified in option word) - an integer specifying minutes after midnight (equivalent to -DEFER in PRIMOS command line.)
12	File size, returned if key is 1.
13-20	(Optional) Logical destination name - must be blank padded (equivalent to -AT on PRIMOS command line). If these words are used, bit 10 of word 3 (option word) must be set to 1.
21-28	(Optional) Substitute filename to be used - must be blank padded (equivalent to -AS on command line). If these words are used bit 11 of word 3 must be set to 1.
29	(Optional) Number of copies (equivalent to -COPIES on command line). If this is used bit 12 of word 3 must be set to 1.

The remaining 11 words are for the extended array. If the extended array is used then bit 16 of info(3) MUST be set.

30	Additional option word. (See below.)
31	Disk number of the SPOOLQ to attach to for the -DISK option. If this word is used bit 1 of word 30 must be set.
32-40	Reserved.

buffer Scratch buffer - this is used to set up control info and to copy the file to the spool queue (key=1).

It must be at least 40 words long. Copy time is inversely proportional to buffer size. Nominal size is between 300 and 2000 words.

buflen Length of buffer.

code Return code (non-zero if error occurred).

The print option word (info(3)) specifies various print(/plot) information and is defined as follows:

Bit	Designation (If set to 1)
1	Fortran format control. (Column 1 contains carriage control information.)
2	Expand compressed listing.
3	Generate line numbers at left margin.
4	Suppress header page.
5	Do not eject page when done.
6	No format control.
7	Plot this file - info(7) specifies words/raster scan.
8	Don't print this file until the time specified in info(11).
9	Print on local printer only — Not used after Rev. 17.
10	Print/plot file at location specified by string in info(13-20).
11	Replace name with string in info(21-28).
12	Spool the number of copies specified in info(29).
13-14	Reserved.
15	Inform user when file has completed printing. Equivalent to -NOTIFY option on command line.
16	Extended array used - MUST be set if words 30 - 40 of info are used.

The extended print option word (info(30)) specifies additional information and is set up as follows:

Bit	Designation (If set to 1)
1	Attach to the SPOOLQ on disk number in info(31).
2	This file is a PRIORITY file. Can only be used in conjunction with the -MODIFY key.
3	Used in conjunction with -MODIFY to remove the PRIORITY attribute from this file.
4	Allow spool$ to place a message in name when code is not 0, designed to be passed as input to errpr$, which is what the spool command has always done.
5-6	Reserved.
7	Truncate all lines to value defined by PROP WIDTH command.
8	Used in conjunction with -MODIFY to remove the DEFER attribute from a file.
9-16	Reserved.

Files That Reside in SPOOLQ

Various files reside in SPOOLQ whose function are sometimes difficult to comprehend. Listed below are explanations that resolve this.

L.FORM: This file contains one form type per line in upper case characters only. If this file is present, it specifies all synonyms for the SPOOL -FORM option; if absent, any form name is allowable.

L.DEST: This file contains one destination name per line in upper case characters only. If this file is present, it specifies all allowable synonyms for the SPOOL -AT option; if absent, any destination name is allowed.

L.DFLT: This file contains only one destination name that becomes the default destination for the SPOOL -AT option. If this file does not exist, the default destination is left blank and a file to be spooled goes to the first printer that matches the FORM subcommand criteria.

L.environment: This file contains one user id per line, in upper case only. It is a list of those users who are permitted to START, STOP, CREATE, MODIFY, and so forth, this particular printer environment. If this file is not present, the file L.USER is checked to see if a user is permitted to control the printer.

L.USER: This file contains one user id per line, in upper case only. It is a list of those users who are permitted to START, STOP, CREATE, MODIFY, and so forth, the printer environments. If this file is not present and there are no L.environment files, only user SYSTEM is permitted to control the printers.

Note

The use of both the L.environment file and the L.USER file allows the System Administrator to globally restrict access to the printers and to then restrict specific printers to specific users with the L.environment file.

L.TYPE: This is a new file for OAS users. It defines the type of printer for each environment file. The information should be entered in upper case as follows:

```
PRINTERNAME1  TYPE
PRINTERNAME2  TYPE
     .         .
     .         .
     .         .
```

Current allowable types are QUME (or 3185), NEC, DIABLO, and KSR. The KSR type is for those users who are printing in MANUAL mode and wish to give the PROP CONTINUE command by entering any character from the keyboard.

NETWORK_INFORMATION.SPOOL: This file consists of one or two ASCII lines. The first line should contain the name of the system on which the SPOOLQ UFD resides. If the SPOOLQ is on a Rev. 18 system, the second line should be REV.18. On Rev. 19 and later systems, no second line is required.

The first line (node name information) is used only by SPOOL -LIST. The text "System:" will be put in front of the first line before the queue is listed but only if entries in the queue will be displayed; that is, there is a list of PRTnnn files to display.

Change of Mode Commands Within the Spooler

Any data line starting with an ^001 (or ^201) causes the spooler to take some action to change control modes. The control mode change is determined by the character(s) following the control character. The character interpretations are as follows:

Mode Control Sequences Within File (Words 1-3 of Record)

WD 1 OCTAL	WD 2 BINARY	
001	-	FF then turn on PAGINATE mode (no page header)
001 000	-	NO-FORMAT mode
001 001	-	Turn on Single FTN control mode (-FTN)
001 002	-	Turn on Double FTN control mode (-COB)
001 004	-	Set PAGINATE mode but do not output page heading or page number.
001 036	data	Use data for new page heading, keep same page number, set PAGINATE mode and perform a FF
001 037	nnnn	Use data word for new PAGINATE page size
000 001	data	Use data for new page heading, reset page number, set PAGINATE mode and perform a FF

Please note that the control sequence "001 004", outlined above, is new at Rev. 19.4.

Format of Accounting Buffer Passed by Spooler Phantom

The subroutine ACCTNG is called by the spooler phantom on completion of printing the file. The calling sequence is:

```
CALL ACCTNG(BUFF)
```

BUFF is a 200 word INTEGER*2 array. The format of this buffer appears to be in a somewhat haphazard order because it has purposely been made compatible with the entries in Q.CTRL. The first 40 words (BUF(1) to BUF(40)) are from the first file entry area in Q.CTRL and the next 122 words (BUF(41) to BUF(162)) are from the file entry's second area in the Q.CTRL file.

Format of Accounting Buffer

BUF(1)	The first 6 characters of user name (ASCII)	6 bytes
BUF(4)	32 character name to be used in banner, usually filename.	32 bytes
BUF(20)	FORM type spooled with.	6 bytes
BUF(23)	Date spooled (as returned by TIMDAT)	6 bytes
BUF(26)	Time spooled in minutes past midnight. (Note that seconds are held in BUF(101))	2 bytes
BUF(27)	Options used	2 bytes

Bit	Designation (If set to 1)
1	Fortran format control (Column 1 contains carriage control information)
2	Expand compressed listing
3	Generate line #'s at left margin
4	Suppress header page
5	Don't eject page when done
6	No format control
7	Plot this file - BUF(33)) specifies words/raster scan
8	Don't print this file until the time specified in BUF(28)
9	Force this file to print on home printer (Not used after rev 17)
10	Print/plot file at location specified by string at BUF(30) (6 chars) + BUF(33) (remaining 10 chars)
11	Spooled with -AS option - banner name in BUF(4)
12	Output the number of copies specified in BUF(39)
13-14	Reserved
15	Notify user when file has completed printing Equivalent to -NOTIFY option on command line
16	Extended array used when calling SPOOL$

BUF(28)	Deferred print time (minutes after midnight) if -DEFER option used.	2 bytes

BUF(29)	Size of file in records.	2 bytes
BUF(30)	First 6 chars of printer printed on	6 bytes
BUF(33)	Raster scan if -PLOT used	2 bytes
BUF(34)	Last 10 chars of printer printed on	10 bytes
BUF(39)	Number of copies requested if -COP option used	2 bytes
BUF(40)	Reserved.	2 bytes
BUF(41)	32 char user name	32 bytes
BUF(57)	Date-time file last modified as returned by RDEN$$. (0 if invalid)	4 bytes
BUF(59)	Length of treename of file spooled	2 bytes
BUF(60)	Treename of file spooled	80 bytes
BUF(100)	Key used in call to SPOOL$, one if copy, two if open, zero if invalid.	2 bytes
BUF(101)	Seconds file spooled. (BUF(27) already used)	2 bytes
BUF(102)	Additional option word if extended array used for call to SPOOL$	2 bytes

Bit	Designation (If set to 1)
1	SPOOLQ to attach to specified by user. Disk number held in BUF(103) (Octal)
2	This file is a PRIORITY file. Can only be used in conjunction with the -MODIFY key.
3	Used in conjunction with -MODIFY to remove the PRIORITY attribute from this file.
4	Allow spool$ to place a message in name when code is not 0, designed to be passed as input to errpr$, which is what the spool command has always done.
5-6	Reserved.
7	Truncate all lines to value defined by PROP WIDTH command.
8	Used in conjunction with -MODIFY to remove the DEFER attribute from a file.
9-16	Reserved.

BUF(103)	Disk number of SPOOLQ to be attached to - 0 if default (first) queue in system.	2 bytes
BUF(104)	Nodename of system spooled from	6 bytes
BUF(107)	Length of nodename in BUF(104)	2 bytes
BUF(108) to	BUF(162) Reserved	110 bytes
BUF(163)	Date & time printing commenced in TIMDAT format	10 bytes
BUF(168)	Date & time printing completed in TIMDAT format	10 bytes
BUF(173)	Printer control word	2 bytes

Bit	Designation (If set to 1)
1	File was 'aborted' during printing.
2	File was 'dropped' during printing.
3	File was 'backed up' during printing.
4	File was 'restarted' during printing.
5-16	Reserved

BUF(174) to	BUF(180) Reserved	14 bytes
BUF(181)	Number of pages printed (including restarts, backups and copies). This is an INTEGER*4 number	4 bytes

BUF(183)	Number of characters printed (including restarts, backups and copies). This is an INTEGER*4 number	4 bytes
BUF(185)	Number of lines printed (including restarts, backups and copies). This is an INTEGER*4 number	4 bytes
BUF(187) to BUF(200)	Reserved	28 bytes

SOFTWARE PROBLEMS FIXED

In FIN mode, lines longer than 132 characters were sent with two extra blank characters, causing problems on printers that have wrap-around capabilities. This has been fixed (SPAR 2000377).

EVFU channel numbers between 15 and 28 are now checked to support the use of EVFU on serial lines. Additionally, setting up EVFU is now performed correctly (SPAR 3001256).

The destination field length in the banner message was fixed (SPAR 3001303).

The EVFU subcommand was fixed to take the command EVFU -NAME ' ' as an error to force the user to give a non-null filename (SPAR 3007947).

The display for EVFU was fixed so that users can differentiate EVFU -ON from EVFU -NAME ON and EVFU -OFF from EVFU -NAME OFF (SPAR 3007942).

Trailing blanks are stripped from output lines. This prevents needless output of characters and, on Printronix printers using plot packages, removes the vertical bars that are the representation of a spaces.

If the phantom is unable to assign the AMLC line at startup, an error message is output and the phantom is logged out.

Characters without parity set were not converted to uppercase. This problem has been fixed.

Formerly, files spooled in -NOFMT mode started on the second line of a page. Files spooled with -NOFMT, -NOHEADER ,-NOEJECT are now printed without a blank line between them.

The problem that caused the last line of each file to print only when the next job started printing was remedied (SPAR 2001904).

The spool phantom now tries for a longer time to open the SPOOLQ>Q.CTRL queues (SPAR 3000619).

The calling sequence to O$ALNN was changed, but not all callers were updated (SPAR 3007883).

SOFTWARE PROBLEMS OUTSTANDING

Users are able to specify control sequences in the destination (SPOOL -AT), alias (SPOOL -AS) and form (SPOOL -FORM) fields. This can cause problems when other users do a SPOOL -LIST (SPAR 3008035).

SYSCOM

NEW FEATURES AND CHANGES

A new key, K$COMO, is added for GPATH$.

PRIMOS II

NEW FEATURES AND CHANGES

Disk controllers at addresses '22 and '23 can now contain paging partitions. These controllers are accessible under PRIMOS II, however, you cannot use MAKE for these controllers under PRIMOS II.

DOCUMENTATION CORRECTION

The following correction should be made to the System Operator's Guide, Volume I.

In Appendix E, page E-3, Step 6. Restart PRIMOS II:

For the P2550, P9650, P9750, P9950, and P9955 the command issued should be DOS instead of BOOT 170000. This is correctly documented in the Prime 2550 Handbook and the Prime 9950 Handbook.

CHAPTER 4

LANGUAGES

UTILITIES

BIND

NEW FEATURES AND CHANGES

BIND is a new linker that produces EPFs. BIND usage is discussed briefly in Chapter 2, EPFs. See also the Programer's Guide to BIND and EPFs.

SEG

NEW FEATURES AND CHANGES

Created dynamic entries (DYNTs) can be as long as 32 characters, although the symbol names are still limited to eight characters.

SOFTWARE PROBLEMS FIXED

CMDSEG now ignores LISTNER_ORDER$ and is no longer required to be run at command level 1 (SPAR 2000638, 2006053, 2002648).

C internal routines are now called properly when reloaded.

Cleanup is inhibited after executing a program.

New storage allocator calls are used for map sorts.

DBG

NEW FEATURES AND CHANGES

DBG now offers full support of EPF programs, that is, programs linked and loaded by BIND. The user should notice no difference in DBG's behavior, as it is indistinguishable from debugging a static mode program, that is, a program linked and loaded by SEG.

To use DBG on a Library EPF, compile the library as a program EPF. When you no longer need to use DBG recompile the program as a library.

Note

When a user invokes DBG on a program that exists both as a .RUN and a .SEG file and does not use the .RUN or .SEG suffix, DBG defaults to the .RUN file.

The C Dollar Extent Operator ($): The dollar extent operator is used to display C arrays of character as a string. It is used in the same manner as the star extent operator. Instead of displaying the entire dimension of the array, however, it displays the array's elements up to the null C character. For more information on the star extent operator, see The Source Level Debugger User's Guide (DOC4033-193). The dollar extent operator is implemented only for C data evaluation.

For C, DBG now supports array operations on pointers. This is useful for evaluating C function arguments that are arrays.

SOFTWARE PROBLEMS FIXED

Certain instances of Pascal abstract-type arrays were not interpreted correctly by DBG. This is now fixed (SPAR 3002646).

DBG now prints the filename involved in a file I/O error in PL1/G programs (SPAR 3002201:).

DBG would crash if one tried to UNWATCH two WATCHLIST entries in a particular order. This is now fixed (SPAR 3005625).

In Pascal, one could not assign or compare the value of a function to an array element. This now works.

Numerous bugs that occurred when using brackets within actionlists have been fixed.

The following apply to the C language:

In C, all bugs with self-referential, recursively-defined structures have been fixed.

C arrays within structures are now evaluated correctly.

C indirection (* operator) from structure members now works correctly.

A C structure pointer operator (->) following a structure member now works corrctly.

Very large C programs containing many user-defined declarations caused DBG to abnormally terminate. This no longer occurs.

Invoking the POINTER (or PTR) function with C data now returns the correct result.

Occasionally, the address of C data containg no bit offset had a bit offset assigned to it. This no longer occurs.

Using the C ARGUMENTS command from within nested braces did not work correctly. This is now fixed.

Using the C ARGUMENTS command on user-defined arguments did not work correctly. This is now fixed.

Labels contained within nested braces were not always found. This is now fixed.

DBG was improperly handling C constants of the form 0.1 or .1. This is now fixed.

The C ARGUMENTS command did not work correctly when used from an inner C block or for user-defined datatypes. Both problems are now fixed.

OUTSTANDING PROBLEMS

The C escape character (\) and its associated special characters are not yet supported. Support is planned (SPAR 3000086).

In C, displaying of UNSIGNED LONG integers with the low order bit on displays a negative number. However, the internal representation and handling is correct. This will be fixed (SPAR 3000013).

PERMANENT RESTRICTIONS

All ARITHMETIC exceptions detected by hardware cause DBG to lose the user program's environment.

DBG stack areas still remain unprotected from being overwritten by a user procedure that, due to its own errors, has developed bad addresses. Users should thoroughly investigate this possibility before assuming that DBG has a bug in it.

Rev. 19.4 DBG does not support operations on Pascal SET datatypes in programs that have been compiled with a pre-Rev. 19.4 Pascal compiler. This is because of the changes and improvments made in Pascal SET handling for Rev. 19.4. This only effects Pascal SETs. All other Pascal datatypes are compatible.

ENVIRONMENT

Rev. 19.4 of DBG requires a Rev. 19.4 or later version of PRIMOS.

INSTALLATION AND BUILD PROCEDURES

Standard.

Note

DBG uses segments 4037 and 4036 and allocates temporary segments not occupied by the user program downward from 4037.

LIBRARIES

Prime's language libraries have been converted to library EPFs. There is a new system directory, LIBRARIES*, in which the libraries are stored.

CBL_LIBRARY

NEW FEATURES AND CHANGES

The COBOL library is now an Executable Program Format (EPF) Library. The .RUN file for the library is the Program Class Library - CBL_LIBRARY in the UFD LIBRARIES*. This library gets referenced when a user issues the Library command LIBRARY CBLLIB with BIND. Users can reference their own libraries by modifying their ENTRY$ search list.

The unshared COBOL library is referenced with the library command LIBRARY NCBLLIB. Both CBLLIB and NCBLLIB can be used with SEG. The files LIB>VCOBLB.BIN and LIB>NVCOBLB.BIN are identical to LIB>CBLLIB.BIN and LIB>NCBLLIB.BIN, respectively.

CCLIB

NEW FEATURES AND CHANGES

The Rev. 19.4 C library introduces the EPF library for C, C_LIB. An EPF library is a major change in functionality compared to previous shared and non-shared static mode libraries. Using the new loader, BIND, C programs can now be loaded much faster.

Some users may wish to continue using SEG. Users who use SEG must continue using CCLIB and cannot switch to C_LIB. Those users will not enjoy the improved performance and reduced loading time that BIND users will notice.

If you use the C EPF library (the recommended practice), you must include the C library in your search rules. The file, SYSTEM>ENTRY$.SR, contains the standard search list for each user. These search rules include the C library, once it has been installed. The following is what SYSTEM>ENTRY$.SR may look like for the average system:

```
LIBRARIES*>TTYIN$.RUN
LIBRARIES*>SYSTEM_LIBRARY.RUN
LIBRARIES*>FORTRAN_IO_LIBRARY.RUN
LIBRARIES*>APPLICATION_LIBRARY.RUN
LIBRARIES*>CC_LIBRARY.RUN
-STATIC_MODE_LIBRARIES
```

For optimal performance, the user should create his own private copy of the search rules by copying them to an ENTRY$.SR file in his own directory and instruct PRIMOS to use this copy of the search rules via the SET_SEARCH_RULE command upon logging in. This file should also be edited so that the CC_LIBRARY.RUN entry occupies the highest place in the search list. In other words, the library that you are likely to call most often should appear at the head of the list. The libraries that you are least likely to call should appear later in the list. Many C users will want to put the C library after the system library. The following is an example of the search list for a typical C user:

```
LIBRARIES*>TTYIN$.RUN
LIBRARIES*>SYSTEM_LIBRARY.RUN
LIBRARIES*>CC_LIBRARY.RUN
LIBRARIES*>FORTRAN_IO_LIBRARY.RUN
LIBRARIES*>APPLICATION_LIBRARY.RUN
-STATIC_MODE_LIBRARIES
```

For Internals Specialists: The dynamic entries (DYNTs) that are formed in order to interface between a C program and the C EPF library contain a CC$ prefixed onto the name of the C library entry. The actual entry in the C EPF library also contains this prefix. For example, PRINTF's DYNT gets changed to CC$PRINTF to match its entry in the C EPF library. The user never sees this through normal use of the C compiler and the C EPF library. The change takes place within BIND upon issuing the LIBRARY C_LIB command. In fact, the only time this appears is if a LINKAGE_FAULT were encountered when trying to dynamically link to an entry in the C EPF library. As long as the C EPF library has been installed and the proper search rules effected by following the above procedures, the user is never aware of this internal naming convention.

Note

This convention is a function of the C EPF library, C_LIB, not the C compiler. Therefore, this does not occur when using the old, unshared library, CCLIB, or when using SEG.

SOFTWARE PROBLEMS FIXED

There are fixes in various library routines to support PRIMOS file units whose number is greater than 128. The C library routines do not support more than 128 files, although there can be 128 binary files (referenced thru READ, WRITE, OPEN, LSEEK, and so forth) and 128 ASCII files (referenced thru FOPEN, FREAD, PUTS, and so forth) (SPAR 3005365).

The floating constant 1.000 is now printed correctly by PRINT$ with %x.zf format (SPAR 3005367).

The routine EXIT now dumps any buffered TTY data before a return to PRIMOS (SPAR 3005369).

The %p format in PRINTF is modified to produce the standard Prime format (SPAR 3005370).

PERMANENT RESTRICTIONS

The C user should be aware that there are conflicts between some of the names in the FORTRAN and System Libraries and those in the C library. One of those routines is EXIT. COBOL and FORTRAN routines, that use STOP or STOP-RUN statements, call EXIT. If a C program were loaded with other languages, the user must be careful that the routine expected to be called is obtained. The C library routines are not compatible with those from the other languages.

To avoid possible conflicts when modules written in C and other languages are mixed within a .RUN or .SEG file, the user should load those routines written in C and the C libraries before those of other languages. To illustrate:

```
[BIND rev 19.4]
: lo c_module_1.bin
: lo c_module_2.bin
   .
   .
   .
: li c_lib                          /* resolves calls to C
                                       specific libraries */
: lo ftn_module_1.bin
: lo ftn_module_2.bin
   .
   .
   .
: li                                /* resolves calls to all
                                       other libraries */
```

The above insures that the C modules call the correct C library routines before attempting to load other modules that call a library routine with a potentially conflicting name.

INSTALLATION AND BUILD PROCEDURES

CCLIB is supplied with CC comprising two directories, CC and CCLIB. To install CCLIB, the user must run the install file, CCLIB.INSTALL.COMI in UFD CCLIB. CCLIB and CC are built separately.

PASCAL_LIBRARY

NEW FEATURES AND CHANGES

The Pascal library is now an Executable Program Format (EPF) Library. The .RUN file for the library is the Program Class Library, PASCAL_LIBRARY in the UFD LIBRARIES*. This library gets referenced when a user issues the Library command LIBRARY PASLIB with BIND. Users can reference their own libraries by modifying their ENTRY$ search list.

The unshared Pascal library is referenced with the Library command LIBRARY NPASLIB. Both PASLIB and NPASLIB can be used with SEG.

PL1G_LIBRARY

NEW FEATURES AND CHANGES

The PL1G library is now an executable program format (EPF) library. The .RUN file for the library is the program class library PL1G_LIBRARY in the UFD LIBRARIES*. This library gets referenced when a user issues the library command LIBRARY PL1GLB with BIND. Users can reference their own libraries by modifying their ENTRY$ search list.

The unshared PL1G library is referenced with the Library command LIBRARY NPL1GLB. Both PL1GLB and NPL1GLB can be used with SEG.

RFTNLIB

SOFTWARE PROBLEMS FIXED

The incorrect exponent overflow problem has been fixed. The R-Mode routine that processes floating point exception interrupts contained a V-Mode instruction which caused the P9950 to go into floating round mode and incorrectly generate an exponent overflow (SPAR 3003012).

SYSTEM_LIBRARY

NEW FEATURES AND CHANGES

The FORTRAN libraries are now Executable Program Format (EPF) Libraries. The .RUN files for the libraries are in the UFD LIBRARIES*. They consist of the Process Class Library, SYSTEM_LIBRARY, and the Program Class Library, FORTRAN_IO_LIBRARY. These libraries are automatically referenced when a user issues the Library command with BIND. Users can reference their own libraries by modifying their ENTRY$ search list.

The new ACTION= clause for the F77 OPEN statement is now supported. The ACTION has to be READ or READ/WRITE to backspace a file with the current implementation.

Dynamic File Units: F77 now supports 141 file units open at the same time. Users should use the following code sequence to translate PRIMOS units to FORTRAN unit numbers instead of computing the numbers.

```
CALL SRCH$$ (K$GETU + K$RDWR + K$NDAM , FILENM(I), NAMELN,
+            FUNIT, FILTYP, CODE)
CALL ATTDEV (LUNIT, PDEV, FUNIT, BUFLEN)
OPEN (UNIT= LUNIT, FILE= FILENM(I), ACCESS= 'DIRECT',
+     FORM= 'FORMATTED' , RECL= 30)
```

LUNIT is the unit number that FORTRAN uses; that is, the OPEN statement. FUNIT is the file unit that PRIMOS uses. The key K$GETU causes SRCH$$ to open the file name, FILENM(I), on an unused file unit selected by PRIMOS, FUNIT.

F77 support now conforms to the ANSI standard for FORTRAN77. This required changes that could be user visible. When the exponent of a number exceeded the exponent field on output, the printing of a = or a $ and the most significant exponent digits were an extension. (Reference FORTRAN77 Reference Guide (DOC4029-183L), page 4-26). The entire field is filled with asterisks as required by the standard. If the width of the exponent field is unspecified and the exponent requires three digits, the E, D, or Q is implied before the sign of the exponent.

The backspacing of a variable length record on a binary disk file is now supported.

SOFTWARE PROBLEMS FIXED

The 'T' FTN carriage control is supported (SPAR 2000105).

These related B-FORMAT problems have been fixed (SPARs 2000537, 2000623, 2001275, 2002652, 2002860, 2002971, 2005126, 3002524, 3002558).

A file protected with 'UR' can be opened with the FORTRAN OPEN statement without an error (SPAR 2005421).

EXP and DEXP do not give a size error when the argument is less than -22713 (SPAR 2004808, 2003037).

A real zero raised to the real zero power returns a diagnostic as suggested by Cody and Waite, Software Manual for the Elementary Functions, Page 87 (SPAR 2001064, 2001220).

TU and TL give correct results when the next T is backwards from the previous T in the FORMAT string (SPAR 2002366).

The routines F$SCIPWR and F$SCCPWR are loaded with the library for a routine compiling with the -INTS option (SPAR 2002292).

The following routines were added as direct entrance calls to the operating system: TEXTO$, CHG$PW, ASNLN$, USER$, LON$CN, and LON$R (SPAR 3001475).

NAMELIST, used to input from the terminal, works correctly both the first time and subsequent times (SPAR 3002057).

Reading into a REAL*8 variable using an F20.2 format with FTN and entering the character string -0, the REAL*8 variable now has the correct value of 0.0 (SPAR 3001490).

The output of a character*1 variable with an even value of 10 or greater correctly indicates overflow with an internal file (SPAR 3002217).

A double precision integer with the smallest negative value of -2147483648 is represented correctly internally in binary (SPAR 3005964).

The smallest double precision negative integer, with the sign bit on and all the other bits off, correctly outputs -2147483648 (SPAR 2001147, 2000122).

F77 B format now works correctly when only one dollar sign precedes a comma (SPAR 2004579).

The Fortran OPEN statement returns to the calling program when IOSTAT is specified without ERR (SPAR 3005895).

In a program class library, modified IPs are now reinitialized correctly after the library is reinvoked (SPAR 3010175).

SOFTWARE PROBLEMS FIXED

Performance Improvement: Performance has been improved for the input and output of single precision, double precision, and quad precision floating point numbers.

The performance and accuracy of the Transcendental Functions has been improved.

INSTALLATION AND BUILD PROCEDURES

The following have changed from SAM files to DAM files which means that MAGRST will not copy the new versions over the old ones:

SPLLIB.BIN
PFTNLB.BIN
NPFTNLB.BIN

VRPG_LIBRARY

NEW FEATURES AND CHANGES

The V-Mode RPG library is now an Executable Program Format (EPF) Library. The .RUN file for the library is the Program Class Library, VRPG_LIBRARY, in the UFD LIBRARIES*. This library gets referenced when a user issues the Library command LIBRARY VRPGLB with BIND. Users can reference their own libraries by modifying their ENTRY$ search list.

The unshared VRPG library is referenced with the Library command LIBRARY NVRPGLB. Both VRPGLB and NVRPGLB can be used with SEG.

The shared FORMS library is now automatically loaded when loading the VRPG library. Therefore, the load of any VRPG program that interfaces with the Prime Forms Management System (FORMS) does not have to load VFORMS.

COMPILERS

BASICV

NEW FEATURES AND CHANGES

LOF (Length Of File): LOF, length of file, is a built-in function. The syntax is L = LOF(channel_number). This function returns the number of records in the direct access file opened on the specified channel number.

ANSI Minimal BASIC Support: The -MIN option provides ANSI minimal BASIC support. The following lists exception handling:

- Division by zero is a non-fatal exception. Machine infinity with sign of the numerator is provided.
- Underflow is a non-fatal exception. Zero is provided.
- An overflow is a non-fatal exception. Positive or negative infinity is provided.
- 0 raised to 0 power is a non-fatal exception. 1 is provided.
- 0 raised to a negative power is a non-fatal exception. Positive machine infinity is provided.
- ON-GOTO control expression greater than the number of line-numbers in the list is a fatal exception.
- Subscript of -1 is a fatal exception.

The following changes in functionality to the existing BASICV apply only if you use the -MIN option.

Expressions: Comparison of string variables and constants are equal if and only if the strings are equal in length and contain identical sequences of characters. Trailing spaces are significant.

INT Function: For negative numbers, INT(x) provides the largest number not greater than x.

ON-GOTO Statement: Expression in an ON-GOTO statement is rounded to the nearest integer.

OPTION BASE Statement Added: Defines default lower bound on array dimensioning to be 0 or 1.

PRINT Statement: The following apply:

- Consecutive commas prints a TAB character (PRINT A,,B)
- Positive numeric output is preceded by a space
- Negative and positive numeric output is followed by a space

Subscripts: Subscripts are rounded to the nearest integer.

TAB Function: The following apply:

- TAB argument is rounded to the nearest integer.
- TAB less than 1 gives an error diagnostic and is replaced by 1.

User-Defined Functions: The following are changes to user-defined functions:

- Reference to an undefined function gives an error.
- A recursively defined function gives an error.

SOFTWARE PROBLEMS FIXED

The ENTER statement destroys the value of a variable from a previous INPUT statement (SPAR 3007556).

An external call to RDTK$$ takes an access violation (SPAR 3001516).

When the MASK 1 option of CVT$$ is used, the parity bit is not turned off for spaces (SPAR 3009204).

The RND function without the RANDOMIZE statement produces an unchanging implementation-defined sequence of pseudo-random numers.

Commas within a quoted string in the INPUT statement are accepted.

CBL

Rev. 19.4.1 CBL follows the 19.3.3 IPR. Additional changes may be made. See the file CBL.RUNO in the INFO directory for complete information.

NEW FEATURES AND CHANGES

The following are new features and changes:

- EXTERNAL files and working storage items are now supported.
- The primary key is no longer restricted to the first characters of a record.

SOFTWARE PROBLEMS FIXED

Figurative constant and all literal moves and compares work incorrectly for fields greater than 256 characters long. (SPAR 2002180, 3001097, 3003753, 3003848, 3003850, 3005780)

Partial key searches work incorrectly on bit string keys (SPAR 2005240).

An empty error file is created when OBSERVATIONS only are generated and the -SILENT option is used (SPAR 2006056).

A move of one element of a 250 element field destroys the field following the table (SPAR 3000262).

An extremely large procedure division aborts compilation with no diagnostic issued (SPAR 3000329, 3001007).

Compound conditional expressions with negated 88s evaluate incorrectly (SPAR 3000330).

A misleading diagnostic is issued for linkage section items referenced within the procedure division but not in the procedure division USING clause (SPAR 3000779, 3002409).

The compiler terminates compilation and leaves T$nnnn open when a COPY library is not found (SPAR 3000784, 3002594).

A STRING statement imbedded in an IF statement assumes a period after the STRING statement (logically terminates the IF) (SPAR 3000815, 3000830).

A COMPUTE with multiplication of non-integers works incorrectly (SPAR 3000827).

No diagnostic is issued for a SEARCH of a non-occurring item (SPAR 3001006).

The user cannot specify the same filenames with SORT when both the USING and the GIVING options are used (SPAR 3001011).

Non-Prime ANSI tapes with odd record lengths are read incorrectly (SPAR 3001034).

COMPUTE truncates the high order digit if the result is larger than s9(6)v99 COMP-3 (SPAR 3001656).

A program with two sorts aborts with SRT# not found (SPAR 3001672).

An IF statement with multiple abbreviations executes incorrectly (SPAR 3001858).

'compute result = field_a / 100' produces a zero result if field_a is defined as COMPUTATIONAL (SPAR 3002348).

An illegal MOVE CORRESPONDING (elementary receiving field) is not diagnosed (SPAR 3002386).

Multiple SET statments in a called program cause values from the first SET statement to be overwritten (SPAR 3002501).

The compiler aborts when processing compound conditionals with subscripts (SPAR 3002519).

DECLARATIVES are not invoked correctly when the I/O verb is part of a conditional (SPAR 3002816).

An item that redefines itself causes the compiler to loop (SPAR 3002828).

The compiler aborts when a multi-dimensional table is referenced and one of the subscripts is an arithmetic expression (SPAR 3002985).

Moving a COMP-1 field to a field with '-' editing symbols works incorrectly (SPAR 3003316).

An index file key defined as a bit string in CREATK works incorrectly (SPAR 3003548).

Numeric editing with zero suppression is not done correctly (SPAR 3003695).

Zero suppression in a table larger than 49,163 characters works incorrectly (SPAR 3003700).

Binary files produced by previous versions of COBOL or CBL execute incorrectly when rewriting with an alternate key (SPAR 3004151).

Compound conditionals cause the compiler to generate erroneous diagnostics (SPAR 3004210).

A 'find_node' internal compiler error occurs for compound conditionals with subscripting (SPAR 3004220).

A START statement on a partial key works incorrectly (SPAR 3004925).

The INSPECT statement with the replacing option replaces data in the field adjacent to the target field when the target is an element of a group item (SPAR 3005157).

A problem was corrected relative to opening PRISAM files when the length of the filename to be opened was less than the dataname of the file (SPAR 3008091).

OUTSTANDING PROBLEMS

In SUBTRACT statements with the CORRESPONDING option, if a data item in one or both of a pair of matched fields contains a 'P' in its picture clause, no more matches will be found.

In MOVE CORRESPONDING statements containing the SIZE ERROR clause, only the first matched pairs have their data moved. If a size error occurs, all receiving fields retain their original data.

INSTALLATION AND BUILD PROCEDURES

CBL is now supplied as two directories: CBL and CBL_LIBRARY. To install CBL, the user must run the following two files:

1. CBL.INSTALL.COMI from CBL
2. CBL_LIBRARY.INSTALL.COMI from CBL_LIBRARY

CC

NEW FEATURES AND CHANGES

CC is Prime's V-mode compiler for the C language. The PRIME C is a true, 32 bit, two pass compiler, that generates object code in the 64V mode format. This release of the compiler contains numerous bug fixes and optimizations and will execute somewhat faster.

C now supports an EPF library named C_LIB. The C EPF library, C_LIB, will not work when loaded with SEG. Use BIND to create .RUN files of programs (instead of .SEG files) and to utilize C_LIB.

Some users may wish to continue using SEG. Users who use SEG must continue using CCLIB and cannot switch to C_LIB. Those users will not enjoy the improved performance and reduced loading time that BIND users will notice.

If you use the C EPF library (the recommended practice), you should read the discussion of the C library in the Libraries section of this chapter.

The following are examples of using BIND to create C programs. Suppose a user wished to load three C routines whose filenames are XX.BIN, YY.BIN, and ZZ.BIN into a C program which would be called XXX.RUN. The command to BIND would be:

```
BIND XXX -lo XX YY ZZ -li C_LIB -li
```

This procedure works very well provided that the main program is not expecting to receive the command arguments in the usual UNIX style. If your program does start with a main program that expects to receive the command arguments from the command line, you must use the LI option to BIND to first load CCMAIN. Remember that in this case, your main program must be named MAIN. The following is an example of the BIND command which would bind the C program if one of the above files has a main routine which took such arguments:

```
BIND XXX -li CCMAIN -lo XX YY ZZ -li C_LIB -li
```

Please note that if you do include CCMAIN in the load list, it must be included first and that the routine which first gets control must be named MAIN. If you did not include CCMAIN, the first routine you load is considered to be the main program and, if it takes arguments, you will probably get a pointer fault.

The above examples illustrate the use of BIND with a single command line performing the entire load. As with SEG, BIND also works interactively by issuing the BIND command. See the appropriate BIND documentation for more information.

The CCLIB file is the unshared library. It is copied to the LIB directory for users who, for some reason, do not wish to use the C EPF library or who wish to continue to use SEG.

Buffered binary update files are now supported by the C library. This is now the default when opening any update file (+ modes). Unbuffered binary update files can still be forced by an explicit call to SETBUF. Use of buffered binary update files is approximately 20 to 30 times faster than unbuffered binary update files.

The compiler now accepts both .C and .CC suffixes for input files. The abbreviations -SOF and -NSOF have been added for -STORE_OWNER_FIELD and -NO_STORE_OWNER_FIELD.

There is added support for the -PBECB and -LBECB compile line options. -LBECB is the default. -PBECB puts the ECBs in the procedure frame. If the user is not using CCMAIN, the main C routine (the first one to be loaded) must not be compiled with -PBECB. Use of -PBECB will enhance the performance of large programs by reducing a program's locality of reference.

SOFTWARE PROBLEMS FIXED

Assignment using an array as an array subscript did not always work correctly (SPAR 3006958).

BIG code for a structure was not automatically generated when a member of that structure caused its size to be greater than a segment (SPAR 3007023).

A bug that caused a byte of precision to be dropped when casting an integer to a pointer_to_a_named_type renaming char (SPAR 3006689).

There was a problem with the generation of external names whose length was longer than eight characters (SPAR 3002494).

A DO statement without the corresponding WHILE terminator had poor error recovery (SPAR 3002496).

Constant casting to a pointer_to_a_function generated spurious error messages (SPAR 3002497).

A character array indexed by a character expression assigned to a character array does not generate the correct code (SPAR 3002498).

External names were generated whose length was longer than eight characters (SPAR 3003148).

The option -EXPLIST no longer overrides a previous -L TTY and forces output to a disk (SPAR 3005371).

The construct ++STRUCT.ARRAY now produces an error rather than a compiler failure (SPAR 3005372).

A problem with the redeclaration of a defined local function in an inner scope was fixed (SPAR 3005373).

A problem involved with the skipping of commented out preprocessor commands was fixed (SPAR 3005374).

The severity of errors concerning extraneous tokens following #IFDEF, #IFNDEF, and #UNDEF was lowered. The processing of an #UNDEF is now completed (SPAR 3005375).

The continuation of define macros now works correctly if the line continuation character immediately follows the macro name (SPAR 3005376).

Code generation for constant indexing into character arrays was improved. This also affects character pointers (SPAR 3005377).

The C program compiled with the -STAT option no longer aborts because of the PL1G library routine P$PUTF not being on the system (SPAR 3007409).

The compiler is now invoked with a .SAVE file rather than .CPL file. This solves a problem with command line processing of equal names. Also, the compiler now responds slightly faster (SPAR 3008371).

The code generated for short and long unsigned divide and modulus has been corrected (SPAR 3008917).

The inefficient code being generated for most character pointer assignments has been corrected. The code generation problem for expressions of the form: A.TEXT[A.LEN] = CHARACTER has also been corrected. Generating an unneeded temporary is no longer used (SPAR 3009227).

The severity of a program with no procedures defined has been lowered to warning (SPAR 3009285).

PERMANENT RESTRICTIONS

All CHAR arguments are promoted to INT arguments when they are passed and they are expected to be used when they are received.

If a constant value is to be passed to a function which expects that parameter to be of type pointer, the constant value should be explicitly cast to type pointer in the function call. The particular case where this is important is in the use of the constant NULL (as defined in STDIO.H) to represent a NULL pointer value. This is because the constant will cause only two 16-bit words to be reserved on the stack for that value, rather than the necessary three.

INSTALLATION AND BUILD PROCEDURES

CC is supplied as two directories, CC and CCLIB. To install CC, the user must run **two** install files, CC.INSTALL.COMI in UFD CC and CCLIB.INSTALL.COMI in UFD CCLIB, and CC.SHARE.COMI in UFD SYSTEM. CC and CCLIB are built separately.

COBOL

> WARNING
>
> SEG should be used as the loader for COBOL programs that use a DECLARATIVE section. Using BIND as the loader may result in incorrect flow of control when declaratives are invoked.

NEW FEATURES AND CHANGES

This section describes the changes in functionality of the old COBOL product.

The COBOL compiler will continue to be released. However, there is now only one set of COBOL libraries (a nonshared and a shared version) that supports both new COBOL (CBL) and old COBOL (COBOL). The libraries are CBLLIB.BIN (shared) and NCBLLIB.BIN (nonshared). The CBLLIB.BIN file is simply a list of DYNTs that are loaded into the run file and resolved at runtime.

For users who do not recompile and reload their programs, the runtime DYNTs will be resolved as always and programs should operate the same.

For users who recompile and reload their programs, either VCOBLB or CBLLIB will satisfy the load sequence as before. It is recommended that users begin to use CBLLIB when loading programsso that users are aware that new COBOL (CBL) is Prime's primary COBOL compiler and is becoming increasingly more so. The VCOBLB.BIN file will continue to exist for customers who do not want to change the application build files. The NVCOBLB.BIN no longer exists because it is replaced by NCBLLIB.BIN; thus users who wish to load their applications with the nonshared library must use NCBLLIB.BIN.

The COBOL directory that contains the run files for the product and is used to install the product has changed. The subdirectory LIB is no longer present and the files COBOL>SYSTEM>(C2014A, C22014B, C4000) have been deleted. The COBOL.INSTALL.COMI file no longer installs the library. The COBOL.SHARE.COMI file no longer shares the library. Installing and sharing of the COBOL library are done by separate procedures. The SHARE command line for the CBLLIB in the system startup file PRIMOS.COMI (or C_PRMO) should be removed.

Basically, the COBOL library is considered as a separate product used by both the old and the new COBOL and comes as a separate package. New COBOL and old COBOL both use this one library.

The COBOL shared library is replaced by an EPF COBOL library.

SOFTWARE PROBLEMS FIXED

Some users experienced UII$ conditions being raised while executing COBOL programs doing MagTape I/O on P9950 machines. This problem has been fixed (SPAR 3002199).

When using the UNSTRING statement without a TALLYING clause on a P9950, some users experienced the UII$ condition. This problem has been fixed (SPAR 3002194).

ENVIRONMENT

The COBOL product uses the EPF libraries. At Revision 19.4, CBLLIB is an EPF. The nonshared library NCBLLIB.BIN is, as before, simply a binary file containing all the runtime code.

F77

NEW FEATURES AND CHANGES

Compiler Options: Changes have been made in the area of compiler options for Rev. 19.4. These changes include new options that have been added to implement new functionality, new syntaxes that are intended to replace older syntax forms of already existing options, new abbreviations of options, and others. These changes were implemented in an effort to make Prime's Common Backend-based translator products more standardized in their user interface among themselves and with other Prime software products. The list that follows this discussion describes all the options that are supported by the compiler at Rev. 19.4, including new options. The other categories of changes are more fully explained in the next few paragraphs.

Note that some of the changes that replace current compiler option functionality or specification imply that the current forms that they replace are now considered obsolete. This means that at some particular revision in the future these older forms will be in error and will not work. Depending on the functionality, this will either be Rev. 20 or Rev. 21. However, the obsolete forms will be supported until then but use of them will cause a warning to be issued by the compiler.

New option syntaxes have been introduced that are intended to replace older forms of some already existing options. For example, the new syntax -OPTimize n, where n is a decimal number that signifies a level of optimization to be performed, is intended to replace the older syntax of the options -OPTIMIZE, -NOOPTIMIZE, and so on. (See below for a fuller discussion of this particular option).

Most currently existing options now have new abbreviations that are intended to replace the older abbreviations, whether they have been documented or not. For example, the older abbreviation for -RANGE has been previously documented to be -R but -RA would also act to turn on range checking. Beginning at Rev. 19.4, each compiler option will have one full specification and, at most, one abbreviation. In the list that follows this discussion, the proper abbreviation for each option is indicated by capitalized letters in the full option specification. As an example, the full specification for turning on range checking is -RAnge and the abbreviation, indicated by the capital letters, is -RA.

Formation of the negated forms of those options that have them (for example, -DEBUG and -NODEBUG) is now done by prefixing the name of the positive form with -No_ (-DeBuG and -No_DeBuG). To form the abbreviated name of a negated option, prefix the positive abbreviated name with -N (in this case, -DBG and -NDBG). The older form of option negation will no longer be supported in the future.

Processing of compiler options has previously allowed such anomalous behavior as duplicate and conflicting options on the same command line

(such as specifying an option twice or specifying -PRODUCTION and -NOPRODUCTION for the same compiler invocation). In the case of conflicting options, the one that had been specified last on the command line was the one that took control. This behavior is now considered obsolete and will no longer be supported beginning at Rev. 20. Note that redundant specification of any source, binary, or listing files has never been supported by Prime's translator products.

Special note must be made about listing files and the use of options that imply that a listing file be produced. To begin with, options that specify that a listing is to be produced (-Listing, -No_Listing or -Listing NO) must be distinguished from those that specify what is to go into a listing if one is produced (such as -XRef, -MAp). If an option like -XRef is specified during a compiler invocation, a listing file is produced in the absence of an explicit -Listing option. However, if -No_Listing happens also to be specified at the same time as -XRef, there is a conflict. Beginning at Rev. 19.4, production of a listing file is based upon a hierarchical order:

1. If -Listing [pathname] or -No_Listing appear on the compiler command line, those switches determine whether a listing is produced or not, independent of the appearance of any options that specify what goes into a listing. This means that -L -EXP produces a listing and that -NL -EXP does not.

2. If -Listing or -No_Listing was not explicitly set and the positive form of an option that controls the contents of the listing is specified, a listing file will be produced. Note that the sole appearance of the negative form of such an option, for example -No_MAp, does not imply a listing and therefore none will be produced.

3. If no options relating to a listing are specified on the command line, a listing is produced only if -Listing is set as the default at a particular installation.

Following is a list of the options supported by the compiler and a brief statement of their purpose. Compiler-supplied default options are noted by an asterisk. The full name of each option is listed along with the name of its opposite, or negated, form if it has one. Proper abbreviations are indicated by capitalized letters. Abbreviations now considered obsolete are listed as such. The initial default settings will be stated for each compiler option. These can be changed at the user's particular installation.

-32I: Generates 32I mode code. No abbreviation. Obsolete: -3 (was documented) ,-32.

-32IX: New. Generates 32I mode code that gives improved performance when accessing large data objects (such as segment-spanning arrays or common blocks). No abbreviation.

Note

A program compiled with -32IX can be run only on a P2550, a P9650, and a P9750 system, or on a P9950 system that has been modified for this purpose, or on any P9955. If it is run on any other Rev. 19.4 system, unimplemented instruction (UII$) faults will occur.

-64V(*): Generates 64V mode code. No abbreviation. Obsolete: -6 (was documented), -64. Default: -64V.

-Allow_PREconnection(*); -No_Allow_PREconnection: New. If this option is specified, preconnection of the listing output to a pre-opened unit 2, or of the binary output to a pre-opened unit 3, is allowed. If the negative form is specified, the compiler always opens and closes the listing and binary files (and will use dynamic file units). Default: -Allow_PREconnection.

-BIG; -No_BIG(*): New: -nbig. Handles array addressing calculations for all arrays passed as arguments as though the array spans segment boundaries. These calculations need to be more complex and therefore execute more slowly. Obsolete: -NOB, -nobi, -nobig. Default: -No_BIG.

-Binary [arg](*); -No_Binary: New: -no_binary, -nb. Specifies binary object file. Obsolete: -bi, -bin, -bina, -binar, -nob, -nobi, -NOBIN, -nobina, -nobinary. In addition, it is now considered obsolete to use arg equal to NO to specify that no binary file is to be produced and equal to YES to indicate that a binary file is to be produced, though this form will be supported for Rev. 19.4 only. This also applies to YES. At subsequent releases, binary files with those names would be created. Use the -No_Binary option instead of -Binary NO. Default: -Binary.

-CLUster: New. This option is used to specify that the source code being compiled constitutes a cluster that is to be optimized together. At Rev. 19.4, this option is taken to imply that the source code is a complete program. At later revisions it will be extended to include libraries of routines. cluster is the term chosen to name a collection of program units that have been compiled together in order to maximize the optimizations that can be performed. Use of this option means that the compiler can make certain assumptions that are relevant to optimization. The compiler checks the validity of these assumptions when possible but the responsibility for their validity rests with the user.

The assumptions are the following:

- The file compiled with the -CLUSTER option is assumed to be a program with a single entry point. If the file has a main program in it, then that is the entrypoint. If the user has used the -MAIN option to specify a main entry procedure, then that routine is the program entrypoint. Otherwise, the first routine is taken to be the program entrypoint.

- The compiler need not make any procedure or data entry points of the cluster visible outside the cluster except the main procedure entrypoint. All other procedures may be QUICK-called or expanded inline. The user should also specify -OPT 4 when using -CLUSTER for this purpose. See the discussion of Optimization Levels in this document. The binary files from such a compilation cannot be combined with other modules that expect to call these procedures.

- At later revisions, this list of assumptions will be expanded as further optimization options are added. For example, support will be added for clusters that are libraries of routines in which some are private and some are externally callable.

-DClvar; -No_DClvar(*): New: -ndc. Flags undeclared variables. Obsolete: -dcl, -dclv, -dclva, -NODC, -nodcl, -nodclv, -nodclva.

-DeBuG; -No_DeBuG(*): New: -ndbg. Generates full debugger (DBG) functionality code. Obsolete: -DE, -DEB, -debu, -NOD, -NODE, -nodeb, -nodebu, -nodebug.

-DOl; -No_DOl(*): Performs one-trip DO loops according to the way FTN performs them. Obsolete: -NODO.

Note

The l is not included in the abbreviations.

-DYnm(*): Allocates local storage dynamically. The opposite of -SAVE. Obsolete: -dyn.

-ERRList; -No_ERRList(*): New: -nerrl. Prints errors-only listing. Obsolete: -errli, -errlis, -NOERRL, -noerrli, -noerrlis, -noerrlist.

-ERRTty(*); -No_ERRTty: New: -no_errtty, -nerrt. Prints error messages at the user terminal. Obsolete: -errtt, -NOE, -NOERRT, -noerrtt, -not, -nott, -NOTTY, -notty, -nottyd, -nottydi, -nottydia, -nottydiag, -nottydiags.

-EXPlist; -No_EXPlist(*): New: -exp, -no_explist, -nexp. Prints listing including assembler-like output. Obsolete: -EX, -expl, -expli, -explis, -NOEX, -NOEXP, -noexpl, -noexpli, -noexplis.

-FRN; -No_FRN(*): New: -no_frn, -nfrn. Generates special code to improve accuracy of single-precision floating-point calculations (Floating RouNd on stores). Obsolete: -F, -fr, -nof, -nofr, -NOFRN.

-FTN_Entry; -No_FTN_Entry(*): New. Means that all procedures passed as actual parameters are to be passed in the FTN way. Use this option only if the routine you are calling is known to be a FTN module. If you use this to call an F77 routine, you will get an error.

-Full_Help: New. This option is similar to the -help option, except that in addition to the usage summary, a description of the meaning of each compiler option is given.

-Full_OPTimize: New. This option is used to ensure that the maximum amount of optimization available is used. This may mean that in succeeding revisions, if more optimizations are available, compilation may slow down. The program listing tells what optimization level is implied by the use of this option. This level may vary from language to language. It is currently set to optimization level 4 for the F77 compiler.

-Help: New. This option will produce information on using the invoked compiler including a list of all compiler options. However, descriptions of the options are not given. The user is referred to the system HELP command to obtain full information about the installed compiler and to the -Full_Help option for full information about the invoked compiler.

If the user makes a mistake in specifying the options or if only the name of the compiler is given on the command line, the user is referred to the -help option after the error message is given.

-Input pathname: This is an alternate way of specifying the source of the input file. If pathname is 'tty', input comes from the user terminal. Obsolete: -in, -inp, -inpu.

-INIL(*): Makes INTEGER default to INTEGER*4.

-INIS: Makes INTEGER default to INTEGER*2.

-LCase: Distinguishes lower and uppercase characters in the source program. Keywords must be in uppercase in F77. Obsolete: -lca, -lcas.

-Listing [file]; -No_Listing(*): New: -no_listing, -nl. This option is used both to specify that a source file listing is to be produced and, optionally, to specify where the listing is to go. Obsolete: -li, -lis, -LIST, -listi, -listin. In addition, several keywords used for the file specification have special meanings. SPOOL specifies that the listing is to go directly into the spool queue; TTY specifies that the listing is to go to the terminal; YES means that a listing is desired; and NO means that a listing is not desired. Note that uses of file equal to YES or NO are now considered obsolete. At the next release, the compiler would create listing files with those names. The -Listing NO form is replaced by the -No_Listing option, and the YES in -Listing YES is redundant.

-LOGL(*): Makes LOGICAL default to LOGICAL*4.

-LOGS: Makes LOGICAL default to LOGICAL*2.

-MAIN program-entry-name: New. This option is used in conjunction with the -CLUSTER option to specify the top level routine that is the main program entry for those cases where there is no main program that is distinguished by the source program.

-MAp(*); -No_MAp: Produce a listing with a map of data and procedure names. Obsolete: -no_m.

-OFFset; -No_OFFset(*): This option causes a map of the code location of each executable statement to be produced in the listing. New: -no_offset, -noff. Obsolete: -o, -of, -offs, -offse, -noof, -nooff, -nooffs, -nooffse, -nooffset.

-OPTimize [dec-integer]: New: the decimal integer. This option is the same as the existing optimize option except that it may be followed by a decimal integer that specifies an optimization level. If the decimal integer is not specified, a default value that depends on the language is supplied. The new option -Full_OPTimize is equivalent to -OPTimize n, where n is the maximum effective level of optimization for the given language being compiled. The default level is -OPT 2 and -Full_OPTimize is optimization level 4. To turn optimization off, -OPT 0 should be used. Note that a space is required between the option (-OPT), and the value. Obsolete: -NO_OPTIMIZE -OP, -OPT1, -OPT2, -OPT3, -NOOP, -NOOPTIMIZE.

Following is a brief description of what types of optimzations are performed at each level. Each optimization level performs all the optimizations of the next lower level plus those that are listed. Note that the functionality associated with some levels at Revision 19.4 may change in the future and that more levels of optimization may be added at later revisions.

0 Perform no optimizations. This level replaces the option -NOOPTIMIZE.

1 Pattern replacement

2 Common subexpression elimination

3 Loop invariant removal

4 Strength reduction of some common operations including indexing of large arrays and elimination of unreachable code

Internally-nested procedures are made quick, that is, called by a Jump to Subroutine instruction rather than a Procedure Call if conditions allow. Basically, the conditions under which a procedure are made quick are that it be called simply, that is, called from one place. For example, procedure C can be quick if it's called from procedure A. But if it's also called from B where B is a separate procedure from A, then C can't be quick.

When -CLUster is specified on the command line for F77 in addition to optimization level 4, all the subroutines in the file being compiled become candidates to be made quick.

-OVerFlow; -No_OVerFlow: New. This option enables integer exception conditions including overflow and division by zero. Default: -No_OVerFlow. Obsolete: -OVE, -OVER, -OVERF, -OVERFL, -OVERFLO, -NOOV, -NOOVE, -NOOVER, -NOOVERF, -NOOVERFL, -NOOVERFLO, -NOOVERFLOW. Default: -No_OVerFlow.

-PBECB; -No_PBECB(*): Generates code to load Entry Control Blocks (ECBs) into the procedure frame. Obsolete: -PB, -pbe, -pbec.

-PRODuction; -No_PRODuction(*): New: -prod, -no_production, -nprod. Generates code for partial debugger functionality. Obsolete: -P, -pr, -pro, -produ, -produc, -product, -producti, -productio, -NOP, -nopr, -nopro, -NOPROD, -noprodu, -noproduc, -noproduct, -noproducti, -noproductio, -noproduction.

-RAnge; -No_RAnge(*): New: -ra, -no_range, -nra. Generates runtime code that checks subscript ranges. Obsolete: -R, -ran, -rang, -NOR, -nora, -noran, -norang, -norange.

-SAve: Allocates local storage statically. The opposite of -DYNM. Obsolete: -sav.

-SIlent [integer]: New with decimal argument. The decimal argument is the severity level such that errors of that severity and less are not reported. If no value is given, a value of 1 is assumed. The default severity level is 0. Note that a space is required between the option and the value. Obsolete: -SIL, -SIL1, -SIL2, -SIL3, -sile, -silen, -SILENT, -SILENT1, -SILENT2, -SILENT3.

-Source pathname: This is an alternate way of specifying the source of the input file. If pathname is 'tty', input comes from the user terminal. Obsolete: -so, -sou, -sour, -sourc.

-SPACE: New. This option specifies that space is to be given preference over runtime speed in optimization consideration. The opposite of -SPACE is -TIME, which means that optimization is to favor runtime speed over space.

-STATistics; -No_STATistics(*): New: -NO_STATISTICS. Displays compilation statistics at the terminal. Obsolete: -STATI, -STATIS, -STATIST, -STATISTI, -STATISTIC, -NOST, -NOSTA, -NOSTAT, -NOSTATI, -NOSTATIS, -NOSTATIST, -NOSTATISTI, -NOSTATISTIC, -NOSTATISTICS, -TO, -TOT, -TOTA, -TOTAL, -TOTALS.

-Store_Owner_Field; -No_Store_Owner_Field(*): Causes the identity of the current routine to be stored in a known place for use by trace back routines. This obsoletes the -NO_OWNERID and -NOOWNERID options of CBL. Obsolete: -STO, -STOR, -STORE, -STORE_, -STORE_O, -STORE_OW, -STORE_OWN, -STORE_OWNE, -STORE_OWNER, -STORE_OWNER_, -STORE_OWNER_F, -STORE_OWNER_FI, -STORE_OWNER_FIE, -STORE_OWNER_FIEL, -NO_STO, -NO_STOR, -NO_STORE, -NO_STORE_, -NO_STORE_O, -NO_STORE_OW, -NO_STORE_OWN, -NO_STORE_OWNE, -NO_STORE_OWNER, -NO_STORE_OWNER_, -NO_STORE_OWNER_F, -NO_STORE_OWNER_FI, -NO_STORE_OWNER_FIE, -NO_STORE_OWNER_FIEL, -NSO, -NO_OW, -NO_OWN, -NO_OWNE, -NO_OWNER, -NO_OWNERI, -NO_OWNERID.

-TIME(*): New. Means optimization is to favor runtime speed over space. The opposite of -TIME is -SPACE, which specifies that space is to be given preference over runtime speed in optimization selection.

-UPcase(*): Map source program to uppercase (except for quoted literals). Obsolete: -U, -UPC, -UPCA, -UPCAS.

-XRef; -No_XRef(*): Produce listing with cross reference of data/procedure names. Obsolete: -X, -XRE, -NOX, -NOXRE, -NOXREF.

Output of Numerical Data: There are some changes in the way that the runtime I/O library edits numerical data on output (via a WRITE or PRINT statement). When the number of characters that represent numerical data exceeds the specified field width or when an exponent exceeds its specified length using the Ew.dEe or Gw.dEd edit descriptors, the entire field is now filled with asterisks. Note that when numbers that contain three-digit exponents are output, the letter E that precedes the exponent is dropped in an attempt to fit the representation into the specified field width before resorting to filling the entire field with asterisks.

This functionality breaks with Prime's traditional approach to this situation. That approach was that a numerical value was right truncated and either a dollar sign ($) or equals sign (=) was output as the first character of the number if the specified field width was insufficient to represent it. However, this new approach conforms with ANSI X3.9-1978 FORTRAN 77.

Floating Point Libraries: The assembly language mathematical routines in the runtime libraries are now built with the -ROUND option. This option causes floating point literals used in these routines to potentially have up to one half bit of increased accuracy. Programs written in F77 may produce different results when they use the libraries' mathematical routines. In most cases, the differences will be improvements in precision. In a small number of cases, cancellation could reveal rounding errors in previous parts of a calculation. Generally, this change ensures better accuracy for many computations with the same number.

OPEN Statement Extension: A Prime extension to the OPEN statement has been implemented for Rev. 19.4 that allows files to be opened for reading only, writing only, or both. The syntax of the new specifier is:

```
ACTION=act
```

act is a character expression whose value is READ, WRITE or READ/WRITE. READ specifies that only input and file positioning operations may be performed on the file opened as such. WRITE specifies that only writing and file positioning operations may be performed. READ/WRITE permits all types of I/O operations. The default action (if ACTION= is omitted from an OPEN statement), as it has been in the past, attempts

to first open the file for both reading and writing and then open the file for either reading or writing if any errors are encountered.

A file opened with the READ attribute allows multiple readers. A file must be explicitly closed and re-opened by the FORTRAN CLOSE and OPEN statements to give it a different ACTION attribute if this is so desired.

Note that a user must have read rights, given by the file system, to BACKSPACE a file opened ACTION=WRITE by F77.

DO Loops: The code to implement FORTRAN-77 DO loops has been changed at this revision in an effort to produce more efficient DO loop control code. DO loops in programs compiled with the -D01 option are not affected; the compiler continues to generate the same code it always has for these loops.

The main differences in the way FORTRAN-77 DO statements are executed beginning at Rev. 19.4 as opposed to the way they were executed in previous revisions of F77 are outlined here:

- The iteration count (number of times to execute the body of the DO loop) is calculated during processing of the DO statement. This count is determined by the following expression:

  INT ((e2-e1+e3)/e3)

  e1, e2 and e3 represent the values of the initial, terminal, and increment parameters of the DO statement, respectively. If the iteration count is zero or less, the body of the DO loop will not be executed (this same behavior was effected previously in a different way than by calculating the iteration count). Note that the increment parameter of a DO statement cannot be zero (0).

- If the type of the DO variable is INTEGER, the expected final value that it will contain upon satisfaction of the DO loop is calculated. This value is used to determine when to terminate a DO loop that is controlled by an INTEGER DO variable. If the type of the DO variable is REAL, this final value is not calculated and determination of satisfaction of the loop is done by the iteration count.

- The code that controls incrementation of the DO variable and termination the DO loop now appears at the bottom of loops rather than at the top. For loops controlled by DO variables of type INTEGER, the value of the DO variable is compared against the expected final value to determine when to terminate the DO loop. For loops controlled by a variable of type REAL, the iteration count is decremented by one and the loop terminates when this count reaches zero (0).

This new approach to implementing FORTRAN-77 DO loops reduces the number of instructions executed during each iteration of a loop in all cases except for some loops controlled by a variable of type REAL. In these cases, though, the number of instructions remains the same. The largest reduction of instructions occurs for loops that have a variable specified for an increment parameter.

DO1 Loops: Though F77 still implements DO loops in programs compiled with the -DO1 option the same way as it always has, some restrictions on such loops have been lifted. The FTN restrictions for the formation of the DO statement are no longer strictly enforced so that any legal FORTRAN-77 DO statement is compilable under the -DO1 option.

It is recommended that adherence to the FTN restrictions be maintained, however, in a conversion from FTN to F77, especially where extended-range DO loops are involved. F77 allows these in programs when compiled with the -DO1 option only when the terminal and increment parameters are scalar variable references or constants. Therefore FTN programs, when converted to F77, do not behave the same if any DO statements have been modified to include expressions as terminal or increment parameters. F77 issues a warning when it encounters such a situation.

SOFTWARE PROBLEMS FIXED

An ACCESS_VIOLATION occurred from F77 when compiling very large program units. The programs submitted with these SPARs were actually too large for F77 to compile and the ACCESS_VIOLATION occurred because of that. A proper error message is now produced by the compiler for this situation (SPAR 2000162, 2004094, 2005239, 3003224).

Business-style formatting now operates correctly (SPAR 2000623, 2001275, 2002652, 2005126, 3004891, 3007167).

F77 generated code that incorrectly indexed multi-dimensional arrays when the array subscripts were of differing lengths, for example, INTEGER*2 and INTEGER*4. This was only a problem when F77 optimized the code and could previously have been worked around by compiling with -NOOPT. The problem has now been fixed (SPAR 2002024).

An incorrect answer was generated when writing to internal files. The problem was due to an improper decision by the optimizer to eliminate a common subexpression and, similar to the above problem, could have been worked around by compiling with -NOOPT. This problem has also now been fixed (SPAR 2002672).

List-directed, implied-do READ of an array now works correctly (SPAR 2005708).

The nesting of $INSERT files is illegal. This is now properly detected with an appropriate error message of Severity 4 (SPAR 2005762).

The compiler now properly allows the legal use of an assumed-size dummy array in an I/O implied DO loop (SPAR 3000154).

CABS intrinsic function yields correct results in all instances (SPAR 3000975).

F77 programs using simple REAL*4 constants no longer get compilation error messages about bad conversions (SPAR 3000976, 3003225, 3005068, 3007484).

Indexing into an array or calculating a substring with a noninteger expression causes a Severity 1 error message to be issued (SPAR 3001565, 3001577, 3001895).

The F77 OPEN statement now allows specification of READ-ONLY access. Please refer to new functionality mentioned earlier in this document (SPAR 3001788).

Cascading error messages for I/O statements with incomplete arguments are now avoided (SPAR 3002058).

The compiler was issuing a severity 4 error message for a certain legal implied DO loop that appeared in a WRITE statement. This problem has been fixed (SPAR 3002329).

F77 no longer aborts with an ERROR 32 when a subroutine statement has an argument that is an assumed size array (SPAR 3002526).

The legal use of an assumed size dummy array in an I/O implied DO loop was incorrectly flagged as an error by F77. The compiler now properly allows this (SPAR 3002549).

The compiler generated incorrect code for an implied DO loop in an I/O statement when the list item was an array of type CHARACTER*(*). This problem has been fixed (SPAR 3002753).

The data statement for one element of an array, using constants, now compiles correctly (SPAR 3003558, 3006316).

Subroutine CHG$PW is now available (SPAR 3003913).

A READ statement with no input list no longer generates an error message (SPAR 3004161).

Attempts to redeclare variables with the same data type and precision are now caught at compile time and a Severity 2 error message is issued (SPAR 3004892).

An OPEN statement with no ERR= clause that has an IOSTAT= clause will no longer abort the program execution (SPAR 3005895).

F77 compilations with several compile-line options should no longer issue ERROR 230 (SPAR 3005924, 3006144, 3006649, 3006968).

A READ statement using list-directed I/O on a character array now works correctly (SPAR 3006049).

The illegal use of two commas in a row to indicate missing parameters to the INQUIRE statement is now detected at compile time (SPAR 3006312).

The I-O routine now processes floating point constants during Internal File Reads the same way it currently processes them during regular reads,that is, they are now rounded not truncated (SPAR 3006409).

The lack of a continuation character for the second line of a FORMAT statement no longer causes the compiler to abort (SPAR 3006502).

Array subscript calculations involving a mixture of integer precision expressions and variables are now evaluated correctly (SPAR 3006503, 3007126).

Logical expressions compiled under 32I-mode are now correctly evaluated (SPAR 3006666, 3007801).

Past incompatibilities in parameter passing between F77 and FIN have been eliminated with the new compile-line option -FIN_ENTRY (SPAR 3006845).

F77 now correctly assigns an integer*2 value to variable in a character*328 function (SPAR 3006876).

Odd elements of character*1 arrays are now properly handled in internal reads and writes (SPAR 3007647).

Logical expressions are now correctly evaluated in 32I mode (SPAR 3007801).

The compiler now detects a common block name with the same symbolic name as an externally-named entity (SPAR 3008293).

If there was a dummy argument when using -OPT 4 and -CLUSTER with assumed-size attributes to a subroutine which could be quick-called, a RESTRICTED_INSTRUCTION could be raised during program execution. This was caused by not allocating space for dope vector information and having the same space used for return processing. It has been corrected.

F77 does not currently generate code that detects integer zero divide at run time. This has been corrected. (SPAR 2003949).

OUTSTANDING PROBLEMS

DBG does not currently recognize F77 alternate entry points (SPAR 2003741, 2001604).

A legal DIMENSION statement causes the compiler to give an erroneous error message when the DIMENSION statement follows an ENTRY statement that immediately follows a SUBROUTINE statement (SPAR 2002923).

ENVIRONMENT

The F77 compiler requires the top level directory LIBRARIES* to contain the file COMMON_ENVELOPE.RUN.

INSTALLATION AND BUILD PROCEDURES

Standard.

CMDNC0>F77.SAVE is no longer needed and should be deleted.

SYSTEM>FT(2046 2047 2052 2067 2322) are no longer needed and should be deleted.

SYSTEM>F77.SHARE.COMI is no longer needed and should be deleted.

Additionally, the lines to share these files should be removed from the system start-up file, PRIMOS.COMI (or C_PRMO).

FTN

NEW FEATURES AND CHANGES

The assembly language mathematical routines in the runtime libraries are now built with the -ROUND option. This option causes floating point literals used in these routines to potentially have up to one half bit of increased accuracy. Programs written in FTN may produce different results when they use the libraries' mathematical routines. In most cases, the differences will be improvements in precision. In a small number of cases, cancellation could reveal rounding errors in previous parts of a calculation. Generally, this change ensures better accuracy for many computations with the same number.

PASCAL

NEW FEATURES AND CHANGES

The runtime execution speed of all SET operations has been improved by more than 50%.

The runtime execution speed of ASCII file I-O operations has been improved by almost 50% (excluding I-O on the data type, CHAR).

The compile line option, -RANGE, now works for enumerated type and subrange, as well as subscript, range violations.

The compile time options -NOOPT1 and -NOOPT3 have been removed.

Compiler Options: Changes have been made in the area of compiler options for Rev. 19.4. These changes include new options that have been added to implement new functionality, new syntaxes that are intended to replace older syntax forms of already existing options, new abbreviations of options, and others. These changes were implemented in an effort to make Prime's Common Backend-based translator products more standardized in their user interface among themselves and with other Prime software products. The list that follows this discussion describes all the options that are supported by the compiler at Rev. 19.4 including new options. The other categories of changes are more fully explained in the next few paragraphs.

Note that some of the changes that replace current compiler option functionality or specification imply that the forms that they replace are now considered obsolete. This means that at some particular revision in the future these older forms will be in error and will not work. Depending on the functionality, this will either be Rev. 20 or Rev. 21. However, the obsolete forms will be supported until then and use of them will cause a warning to be issued by the compiler.

New option syntaxes have been introduced that are intended to replace older forms of some already existing options. For example, the new syntax -OPTimize n, where n is a decimal number that signifies a level of optimization to be performed, is intended to replace the older syntax of the options -OPTIMIZE, -NOOPTIMIZE, and so forth. (See below for a fuller discussion of this particular option.) The older form of specification will no longer be supported beginning at Rev. 21.

Most currently existing options now have new abbreviations that are intended to replace the older abbreviations, whether they have been documented or not. For example, the older abbreviation for -RANGE has been previously documented to be -R but -RA would also act to turn on range checking. Beginning at Rev. 19.4, each compiler option has one full specification and at most one abbreviation. In the list that follows this discussion, the proper abbreviation for each option is indicated by capitalized letters in the full option specification. As

an example, the full specification for turning on range checking is -RAnge and the abbreviation, indicated by the capital letters, is -RA. The older abbreviations will no longer be supported beginning at Rev. 21.

Formation of the negated forms of those options that have them (such as -DEBUG and -NODEBUG) is now done by prefixing the name of the positive full specification with -No_ (-DeBuG and -No_DeBuG). To form the negation of an abbreviated option, prefix the positive abbreviated name with -N (in this case, -DBG and -NDBG). The older form of option negation will no longer be supported beginning at Rev. 21. Note, however, that the option forms -Listing NO and -Binary NO will still be retained for compatibility reasons, though a note will be issued by the compiler encouraging the use of the newer forms -No_Listing and -No_Binary.

Processing of compiler options has previously allowed such anomalous behavior as duplicate and conflicting options on the same command line (for example, specifying the same option twice or specifying options like -PRODUCTION and -NOPRODUCTION during the same compiler invocation). In the case of conflicting options, the one that appeared last on the command line was the one that took control. This behavior is now considered obsolete. Note that redundant specification of any of the source, binary, or listing files has never been supported by Prime's translator products.

Special note must be made about listing files and the use of options that imply that a listing file be produced. To begin with, options that specify that a listing is to be produced (-Listing, -No_Listing or -Listing NO) must be distinguished from those that specify what is to go into a listing if one is produced (such as -XRef, -MAp). If an option like -XRef is specified during a compiler invocation, a listing file is produced in the absence of an explicit -Listing option. However, if -No_Listing happens also to be specified at the same time as -XRef, there is a conflict. Beginning at Rev. 19.4, production of a listing file is based upon a hierarchical order:

1. If -Listing [pathname] or -No_Listing appears on the compiler command line, that switch determines whether or not a listing is produced independent of the appearance of any options that specify what goes into a listing. This means that -L -EXP produces a listing, and that -NL -EXP does not.

2. If -Listing or -No_Listing was not explicitly set and if the positive form of an option that controls the contents of the listing appears, a listing file is produced. Note that the sole appearance of the negative form of such an option, for example -No_MAp, does not imply a listing and therefore none is produced.

3. If no options relating to a listing were specified on the command line, a listing is produced only if -Listing is set as a default at a particular installation.

Following is a list of the options supported by the compiler with a brief description of each. The full name of each option is listed along with the name of its opposite, or negated, form if it has one. Proper abbreviations are indicated by capitalized letters. Abbreviations now considered obsolete are listed as such. The initial default settings will be stated for each compiler option. These can be changed at the user's particular installation.

-64V: Generates 64V mode code. No abbreviation. Obsolete: -6 (was documented), -64. Default: -64V.

-Allow_PREconnection; -No_Allow_PREconnection: New. If this option is specified, preconnection of the listing output to a pre-opened unit 2 or of the binary output to a pre-opened unit 3 is allowed. If the negative form is specified, the compiler always opens and closes the listing and binary files and uses dynamic file units. Default: -Allow_PREconnection.

-BIG; -No_BIG: New: -NBIG. Handles arrays that span segment boundaries in situations where the compiler cannot tell from the program itself that segment boundaries are spanned. Obsolete: -NOB, -NOBI, -NOBIG. Default: -No_BIG.

-Binary [arg]; -No_Binary: New: -NO_BINARY, -NB. Specifies binary object file. Obsolete: -BI, -BIN, -BINA, -BINAR, -NOB, -NOBI, -NOBIN, -NOBINA, -NOBINARY. In addition, it is obsolete to use arg equal to NO to specify that no binary file is to be produced and equal to YES to indicate that a binary file is to be produced, though this form will be supported indefinitely. Use the -No_Binary option instead of -Binary NO. Default: -Binary.

-COPy; -No_COPy: -COPY means that, when constants are passed by reference, copies of the constants are made to prevent them from being changed if the called procedure modifies that parameter. -NO_COPY means that a copy need not be performed, presumably because the programmer has been careful not to pass constants as actual parameters to formals that get modified. Default: -COPy.

-DeBuG; -No_DeBuG: New: -NDBG. Generates full debugger (DBG) functionality code. Obsolete: -DE, -DEB, -DEBU, -NOD, -NODE, -NODEB, -NODEBU, -NODEBUG. Default: -No_DeBug.

-ERRList; -No_ERRList: New. Prints errors-only listing. Default: -No_ERRList.

-ERRTty; -No_ERRTty:f New: -NO_ERRTTY, -NERRT. Prints error messages at a user terminal. Obsolete: -ERRTT, -NOE, -NOERRT, -NOERRTT, (CBL) -NOT, -NOTT, -NOTTY, -NOTTY, -NOTTYD, -NOTTYDI, -NOTTYDIA, -NOTTYDIAG, -NOTTYDIAGS. Default: -ERRTty.

-EXPlist; -No_EXPlist: New: -EXP, -NO_EXPLIST, -NEXP. Prints listing including assembler-like output. Obsolete: -EX, -EXPL, -EXPLI, -EXPLIS, -NOEX, -NOEXP, -NOEXPL, -NOEXPLI, -NOEXPLIS. Default: -No_EXPlist.

-EXTernal; -No_EXTernal: New: -NEXT. Allows object file to be linked to and from other Pascal procedures and functions. Obsolete: -EXTE, -EXTER, -EXTERN, -EXTERNA, -NOEXT, -NOEXTE, -NOEXTER, -NOEXTERN, -NOEXTERNA, -NOEXTERNAL.

-FRN; -No_FRN: New: -NO_FRN, -NFRN. Generates special code to improve accuracy of single-precision floating-point calculations (Floating RouNd on stores). Obsolete:-F, -FR, -NOF, -NOFR, -NOFRN. Default: -No_FRN.

-Full_Help: This option is similar to the -HELP option except that, in addition to the usage summary, a description of the meaning of each compiler option is given.

-Full_OPTimize: New. This option is used to ensure that the maximum amount of optimization available is used. This may mean that, in succeeding revisions, if more optimizations are available, compilation may slow down. The program listing tells what optimization level is implied by the use of this option. This level may vary from language to language. Full_OPTimize is equivalent to -OPT 3 at Rev. 19.4.1. Default: -OPT 3.

-Help: This option produces information on using the invoked compiler including a list of all compiler options. However, descriptions of the options are not given. The user is referred to the system HELP command to obtain full information about the installed compiler and to the -Full_Help option for full information about the invoked compiler.

If the user makes a mistake in specifying the options or if the name of the compiler only is given on the command line, the user will be referred to the -HELP option after the error message is given.

-Input pathname: This is an alternate way of specifying the source of the input file. If pathname is 'TTY', input comes from the user terminal. Obsolete: -IN, -INP, -INPU.

-Listing [arg]; -No_Listing: New: -NO_LISTING, -NL. This option is used both to specify that a source file listing is to be produced and, optionally, to specify where the listing is to go. Obsolete: -LI, -LIS, -LIST, -LISTI, -LISTIN. In addition, several keywords used for the arg specification have special meanings. SPOOL specifies that the listing is to go directly into the spool queue; TTY specifies that the listing is to go to the terminal; YES means that a listing is desired; and NO means that a listing is not desired. Note that uses of arg equal to YES or NO are now considered obsolete (though they will be supported indefinitely). The -Listing NO form is replaced by the -No_Listing option and the YES in -Listing YES is redundant.

-MAp; -No_MAp: Produce a listing with a map of data and procedure names. Obsolete: -NO_M. Default: -No_MAp.

-OFFset; -No_OFFset: This option causes a map of the code location of each executable statement to be produced in the listing. New: -NO_OFFSET, -NOFF. Obsolete: -O, -OF, -OFFS, -OFFSE, -NOOF, -NOOFF, -NOOFFS, -NOOFFSE, -NOOFFSET.

-OPTimize [dec-integer]: New: the decimal integer. This option is the same as the existing optimize option except that it may be followed by a decimal integer that specifies an optimization level. If the decimal integer is not specified, a default value is supplied. The new option -Full_OPTimize is equivalent to -OPTIMIZE n, where n is the maximum effective level of optimization. To turn optimization off, -OPT 0 should be used. Note that a space is required between the option and the value. Obsolete: -OP, -OPT1, -OPT2, -OPT3, -NOOP, -NOOPTIMIZE -NO_OPTIMIZE.

Following is a brief description of what types of optimzations are performed at each level. Each optimization level performs all the optimizations of the next lower level plus those that are listed. Note that the functionality associated with some levels at Rev. 19.4 may change in the future and that more levels of optimization may be added at later revisions.

0 Perform no optimizations. This level replaces the option -NOOPTIMIZE.

1 Pattern replacement

2 Common subexpression elimination

3 Loop invariant removal

-PRODuction; -No_PRODuction: New: -NO_PRODUCTION, -NPROD. Generates code for partial debugger functionality. Obsolete: -P, -PR, -PRO, -PRODU, -PRODUC, -PRODUCT, -PRODUCTI, -PRODUCTIO, -NOP, -NOPR, -NOPRO, -NOPROD, -NOPRODU, -NOPRODUC, -NOPRODUCT, -NOPRODUCTI, -NOPRODUCTIO, -NOPRODUCTION.

-RAnge; -No_RAnge: New: -RA, -NO_RANGE, -NRA. Generate runtime code that checks subscript ranges. Obsolete: -R, -RAN, -RANG, -NOR, -NORA, -NORAN, -NORANG, -NORANGE.

-SIlent [dec]: New with decimal argument. The decimal argument is the severity level such that errors of that severity and less are not reported. If no value is given, then a value of 1 is assumed. Note that a space is required between the option and the value. Obsolete: -SIL, -SIL1, -SIL2, -SIL3, -SILE, -SILEN, -SILENT, -SILENT1, -SILENT2, -SILENT3.

-Source pathname: This is an alternate way of specifying the source of the input file. If pathname is 'TTY', input comes from the user terminal. Obsolete: -SO, -SOU, -SOUR, -SOURC.

-SPACE: New. This option specifies that space is to be given preference over runtime speed in optimization consideration. The opposite of -SPACE is -TIME, which means that optimization is to favor runtime speed over space.

-STANdard; -No_STANdard: New: -NO_STANDARD, -NSTAN. Generates warning for variance from appropriate standard(s). Obsolete: -STAND, -STANDA, -STANDAR, -NOSTAN, -NOSTAND, -NOSTANDA, -NOSTANDAR, -NOSTANDARD.

-STATistics; -No_STATistics: New: -NO_STATISTICS. Displays compilation statistics at terminal. Obsolete: -STATI, -STATIS, -STATIST, -STATISTI, -STATISTIC, -NOST, -NOSTA, -NOSTAT, -NOSTATI, -NOSTATIS, -NOSTATIST, -NOSTATISTI, -NOSTATISTIC, -NOSTATISTICS, -TO, -TOT, -TOTA, -TOTAL, -TOTALS.

-Store_Owner_Field; -No_Store_Owner_Field: Causes the identity of the current routine to be stored in a known place for use by trace-back routines. This obsoletes the -NO_OWNERID and -NOOWNERID options of CBL. Obsolete: -STO, -STOR, -STORE, -STORE_, -STORE_O, -STORE_OW, -STORE_OWN, -STORE_OWNE, -STORE_OWNER, -STORE_OWNER_, -STORE_OWNER_F, -STORE_OWNER_FI, -STORE_OWNER_FIE, -STORE_OWNER_FIEL, -NO_STO, -NO_STOR, -NO_STORE, -NO_STORE_, -NO_STORE_O, -NO_STORE_OW, -NO_STORE_OWN, -NO_STORE_OWNE, -NO_STORE_OWNER, -NO_STORE_OWNER_, -NO_STORE_OWNER_F, -NO_STORE_OWNER_FI, -NO_STORE_OWNER_FIE,

-NO_STORE_OWNER_FIEL, -NSO, -NO_OW, -NO_OWN, -NO_OWNE, -NO_OWNER, -NO_OWNERI, -NO_OWNERID. Default: -Store_Owner_Field.

-TIME: New. Means optimization is to favor runtime speed over space. The opposite of -TIME is -SPACE, which specifies that space is to be given preference over runtime speed in optimization selection. Default: -TIME.

-UPcase: Map source program to uppercase (except for quoted literals). Obsolete: -U, -UPC, -UPCA, -UPCAS.

-XRef; -No_XRef: Produce listing with cross reference of data/procedure names. Obsolete: -X, -XRE, -NOX, -NOXRE, -NOXREF.

SOFTWARE PROBLEMS FIXED

The functions SUCC and PRED now generate an error for out of range results if compiled with -RANGE. Variables out of range are caught at run time while constants out of range are caught at compile time whether or not -RANGE is used (SPAR 2000353,2000905).

Assigning an illegal value to a subrange type now generates an error if compiled with -RANGE (SPAR 2000354, 3004739, 3004401).

Using the name of an enumerated type as part of the enumeration list now gives a compile time error message (SPAR 2004367).

Set operations have been improved by 50% (SPAR 2005060).

The occurrence of intermittent and erroneous error messages regarding files of INTERACTIVE type has been corrected (SPAR 3000388, 2005528).

An external switch set just following the program heading in a main program is now processed correctly (SPAR 3001335).

Set computations have been improved by 50% (SPAR 3001338).

The debugger can now correctly understand the PASCAL construct ARRAY [CHAR] of ... (SPAR 3001339).

Structures which span segments are now handled correctly (SPAR 3001930).

A case statement based on logical comparisons of chars works correctly now (SPAR 3002610).

The functionality of the built-in function, UNSTR, now matches the documentation. In addition, error messages are now delivered for illegal uses of STRINGs and UNSTRs in built in functions (SPAR 3003798).

Violations of SUBRANGE can now be detected for variables if the -RANGE compile line option is used. The use of constants which violate SUBRANGE bounds are caught at compile time (SPAR 3004401).

A named constant may now have the value -2147483648 (SPAR 3004061).

An element of an array of STRINGs declared inside a record may be written out (SPAR 3004414).

An extra left parentheses in the argument list of a procedure call is ignored and an attempt is made to correct the error.

An external subprogram with any external switches other than the initial one results in an error message.

The compiler now requires that a function's body contain an assignment to the function.

The compiler now prohibits assigning or passing by value objects that are either files or structures containing files.

The compiler now prohibits the use of a FOR statement control variable as an argument to READ within the body of the FOR statement.

The compiler now prohibits the use of a formal parameter as a FOR statement control variable.

The compiler now prohibits passing a FOR statement control variable as a VAR parameter from within the FOR statement.

The compiler now prohibits the modification of a FOR statement control variable within the FOR statement.

The compiler now requires that nested FOR statements have unique control variables.

The compiler now requires that arguments to TRUNC and ROUND be of type REAL or LONGREAL, as the standard specifies. Arguments to TRUNC and ROUND that are of type INTEGER or LONGINTEGER cause severity 1 errors while arguments that are of other illegal types cause severity 3 errors.

The compiler now requires that the body of a CASE statement be nonempty.

The compiler now prohibits SET types of base type STRING.

OUTSTANDING PROBLEMS

Integer expressions whose results are greater than maxint are not processed as longintegers, regardless of the data type they are being assigned to (SPAR 3005126).

Erroneous severity 1 warnings are issued about type conversions even when no conversions are necessary (SPAR 3005127).

PERMANENT RESTRICTIONS

Sparse CASE statements will have longer compile times than non-sparse CASE statements. The compile time is linear with respect to the sparsity.

ENVIRONMENT

Standard.

COMMON_ENVELOPE.RUN is required to run the PASCAL compiler.

INSTALLATION AND BUILD PROCEDURES

Standard. PASCAL is now supplied as two directories: PASCAL and PASCAL_LIBRARY. In order to use PASCAL, you must run two install files, PASCAL.INSTALL.COMI from the UFD PASCAL and, from the UFD PASCAL_LIBRARY, PASCAL_LIBRARY.INSTALL.COMI.

Note

The following files are not needed at Rev. 19.4 and, if they exist, they should be deleted:

```
CMDNC0>PASCAL.SAVE
SYSTEM>PASCAL.SHARE.COMI
SYSTEM>PA2064, PA2065, PA2066, PA2141, PA2147
```

The lines to share PASCAL in the system start-up file PRIMOS.COMI (or C_PRMO) should also be deleted.

PL1G

NEW FEATURES AND CHANGES

On loc condition: The ON ERROR statement is now used to catch the raised condition when a locked record of a MIDAS file is referenced. The corresponding value returned by bif oncode() is 1130 and it can be used to distinguish the special condition from the other conditions.

Compiler Options: Changes have been made to compiler options for Rev. 19.4. These changes include new options that have been added to implement new functionality, new syntaxes that are intended to replace older syntax forms of already existing options, new abbreviations of options, and others. These changes were implemented in an effort to make Prime's Common Backend-based translator products more standardized in their user interface among themselves and with other Prime software products. The list that follows this discussion describes all the options that are supported by PL1G at Rev. 19.4 including new options. The other categories of changes are more fully explained below.

Note that some of the changes that replace current PL1G compiler option functionality or specification imply that the forms that they replace are now considered obsolete. This means that at some particular revision in the future these older forms will be in error and will not work. However, the obsolete forms will be supported until then and use of them will cause a warning to be issued by the compiler.

New option syntaxes have been introduced that are intended to replace older forms of some already existing options. For example, the new syntax -OPTimize n, where n is a decimal number that signifies a level of optimization to be performed, is intended to replace the older syntax of the options -OPTIMIZE, -NOOPTIMIZE, and so forth. (See below for a fuller discussion of this particular option.) The older form of specification will no longer be supported at some time in the future.

Most currently existing options now have new abbreviations that are intended to replace the older abbreviations, whether they have been documented or not. For example, the older abbreviation for -RANGE has been previously documented to be -R but -RA would also act to turn on range checking. Beginning at Rev. 19.4, each compiler option has one full specification and at most one abbreviation. In the list that follows this discussion, the proper abbreviation for each option is indicated by capitalized letters in the full option specification. As an example, the full specification for turning on range checking is -RAnge, and the abbreviation, indicated by the capital letters, is -RA. The older abbreviations will no longer be supported in the future.

Formation of the negated forms of those options that have them (such as -DEBUG and -NODEBUG) is now done by prefixing the name of the positive full specification with -No_ (-DeBuG and -No_DeBuG). To form the negation of an abbreviated option, prefix the positive abbreviated name

with -N (in this case, -DBG and -NDBG). The older form of option negation will no longer be supported in the future. Note, however, that the option forms -Listing NO and -Binary NO will still be retained for compatibility reasons, though a note will be issued by the compiler encouraging the use of the newer forms -No_Listing and -No_Binary.

Processing of compiler options has previously allowed such anomalous behavior as duplicate and conflicting options on the same command line (for example, specifying the same option twice or specifying options like -PRODUCTION and -NOPRODUCTION during the same compiler invocation). In the case of conflicting options, the one that appeared last on the command line was the one that took control. This behavior is now considered obsolete and will no longer be supported in the future. Note that redundant specification of any of the source, binary, or listing files has never been supported by Prime's translator products.

Special note must be made about listing files and the use of options that imply that a listing file be produced. To begin with, options that specify that a listing is to be produced (-Listing, -No_Listing or -Listing NO) must be distinguished from those that specify what is to go into a listing if one is produced (such as -XRef, -MAp). If an option like -XRef is specified during a PL1G invocation, a listing file will be produced in the absence of an explicit -Listing option. However, if -No_Listing happens also to be specified at the same time as -XRef, there is a conflict. Beginning at Rev. 19.4, production of a listing file is based upon a hierarchical order:

1. If -Listing [pathname] or -No_Listing appears on the compiler command line, this switch determines whether or not a listing is produced independent of the appearance of any options that specify what goes into a listing. This means that -L -EXP produces a listing and that -NL -EXP does not.

2. If -Listing or -No_Listing was not explicitly set and if the positive form of an option that controls the contents of the listing appears, a listing file is produced. Note that the sole appearance of the negative form of such an option, for example -No_MAp, does not imply a listing and none is produced.

3. If no options relating to a listing were specified on the command line, a listing is produced only if -Listing is set as a default at a particular installation.

Following is a list of the options supported by PL1G with a brief description of each. The full name of each option is listed along with the name of its opposite, or negated, form if it has one. Proper abbreviations are indicated by capitalized letters. Abbreviations now considered obsolete are listed as such. The initial default settings will be stated for each compiler option. These can be changed at the user's particular installation.

-64V: Generates 64V mode code. No abbreviation. Obsolete: -6 (was documented), -64. Default: -64V.

-Allow_PREconnection; -No_Allow_PREconnection: If this option is specified, then preconnection of the listing output to a pre-opened unit 2 or of the binary output to a pre-opened unit 3 is allowed. If the negative form is specified, PL1G always opens and closes the listing and binary files (and uses dynamic file units). Default: -Allow_PREconnection.

-BIG; -No_BIG: Handles arrays that span segment boundaries in situations where the compiler cannot tell from the program itself that segment boundaries are spanned. Obsolete: -NOB, -NOBI, -NOBIG. Default: -No_BIG.

-Binary [arg]; -No_Binary: New: -NO_BINARY, -NB. Specifies binary object file. Obsolete: -BI, -BIN, -BINA, -BINAR, -NOB, -NOBI, -NOBIN, -NOBINA, -NOBINARY. In addition, it is now considered obsolete to use arg equal to NO to specify that no binary is to be produced and equal to YES to indicate that a binary file is to be produced, though this form will be supported indefinitely. Use the -No_Binary option instead of -Binary NO. Default: -Binary.

-COPy; -No_COPy: -COPY means, when constants are passed by reference, copies of the constants are made to prevent them from being changed if the called procedure modifies that parameter. -NO_COPY means that such copy need not be performed, presumably because the programmer has been careful not to pass constants as actual parameters to formals that get modified. Default: -COPy.

-DeBuG; -No_DeBuG: New: -NDBG. Generates full debugger (DBG) functionality code. Obsolete: -DE, -DEB, -DEBU, -NOD, -NODE, -NODEB, -NODEBU, -NODEBUG. Default: -No_DeBug.

-ERRList; -No_ERRList: New: -ERRLIST, -NERRL. Prints errors-only listing. Obsolete: -ERRLI, -ERRLIS, -NOERRL, -NOERRLI, -NOERRLIS, -NOERRLIST. Default: -No_ERRList.

ERRTty; -No_ERRTty: New: -ERRTTY, -NO_ERRTTY, -NERRT. Prints error messages at user terminal. Obsolete: -ERRTT, -NOE, -NOERRT, -NOERRTT, (CBL) -NOT, -NOTT, -NOTTY, -NOTTY, -NOTTYD, -NOTTYDI, -NOTTYDIA, -NOTTYDIAG, -NOTTYDIAGS. Default: -ERRTty.

-EXPlist; -No_EXPlist: New: -EXP, -NO_EXPLIST, -NEXP. Prints listing including assembler-like output. Obsolete: -EX, -EXPL, -EXPLI, -EXPLIS, -NOEX, -NOEXP, -NOEXPL, -NOEXPLI, -NOEXPLIS. Default: -No_EXPlist.

-FRN; -No_FRN: New:-NO_FRN, -NFRN. Generates special code to improve accuracy of single-precision floating-point calculations (Floating RouNd on stores). Obsolete:-F, -FR, -NOF, -NOFR, -NOFRN. Default: -No_FRN.

-Full_Help: This option is similar to the -HELP option, except that, in addition to the usage summary, a description of the meaning of each compiler option is given.

-Full_OPTimize: New. This option is used to ensure that the maximum amount of optimization available is used. This may mean that, in succeeding revisions if more optimizations are available, compilation may slow down. The program listing will tell what optimization level is implied by the use of this option. This level may vary from language to language. Default: The full optimization level is 3 at Rev. 19.4.1.

-Help: New. This option produces information on using the PL1G compiler, including a list of all compiler options. However, descriptions of the options are not given. The user is referred to the system HELP command to obtain full information about the installed compiler and to the -Full_Help option for full information about the invoked compiler.

If the user makes a mistake in specifying the options or if only the name of the compiler is given on the command line, the user will be referred to the -HELP option after the error message is given.

-Input pathname: This is an alternate way of specifying the source of the input file. If pathname is 'TTY', input comes from the user terminal. Obsolete: -IN, -INP, -INPU.

-LCase: Distinguishes lower case and upper case characters in the source program. System calls must be in upper case. Obsolete: -LCA, -LCAS.

-Listing [arg]; -No_Listing: New: -NO_LISTING, -NL. This option is used both to specify that a source file listing is to be produced and, optionally, to specify where the listing is to go. Obsolete: -LI, -LIS, -LIST, -LISTI, -LISTIN. In addition, several keywords used for the arg specification have special meanings. SPOOL specifies that the listing is to go directly into the spool queue. TTY specifies that the

listing is to go to the terminal. YES means that a listing is desired and NO means that a listing is not desired. Note that uses of arg equal to YES or NO are now considered obsolete (though they will be supported indefinitely). The -Listing NO form is replaced by the -No_Listing option, and the YES in -Listing YES is redundant.

-MAp, -No_MAp: New. Produce a listing with a map of data and procedure names. Obsolete: -NO_M. Default: -No_MAp.

-NEsting; -No_NEsting: New: -NO_NESTING, -NNE. Adds nesting level numbers in program listing. Obsolete: -NONE, -NONES, -NONEST, -NONESTI, -NONESTIN, -NONESTING. Default: -No_NEsting. Default: -No_NEsting.

-OFFset; -No_OFFset: This option causes a map of the code location of each executable statement to be produced in the listing. New: -NO_OFFSET, -NOFF. Obsolete: -O, -OF, -OFFS, -OFFSE, -NOOF, -NOOFF, -NOOFFS, -NOOFFSE, -NOOFFSET. Default: -No_OFFset. Default: -No_OFFset.

-OPTimize [dec-integer]: New: the decimal integer. This option is the same as the existing optimize option except that it may be followed by a decimal integer that specifies an optimization level. If the decimal integer is not specified, the default value 0 is supplied. The new option -Full_OPTimize is equivalent to -OPTimize n, where n is the maximum effective level of optimization for PL1G. To turn optimization off, -OPT 0 should be used. Note that a space is required between the option and the value. Obsolete: -OP, -OPT1, -OPT2, -OPT3, -NOOP, -NOOPTIMIZE -NO_OPTIMIZE.

Following is a brief description of what types of optimzations are performed at each level. Each optimization level performs all the optimizations of the next lower level plus those that are listed. Note that the functionality associated with some levels at Rev. 19.4 may change in the future and that more levels of optimization may be added.

0 Perform no optimizations. This level replaces the option -NOOPTIMIZE.

1 Pattern replacement

2 Common subexpression elimination (default optimization level)

3 Loop invariant removal

Internally-nested procedures are made quick,that is, called by a Jump to Subroutine instruction rather than a Procedure Call if conditions allow. Basically, the conditions under which a procedure is made quick are that it be called simply, that is, called from one place. For example, procedure C can be quick if it's called from procedure A. But if it's also called from B where B is a separate procedure from A, then C can't be quick.

-OVerFlow; -No_OVerFlow: New. This option enables integer overflow conditions. Obsolete: -OVE, -OVER, -OVERF, -OVERFL, -OVERFLO, -NOOV, -NOOVE, -NOOVER, -NOOVERF, -NOOVERFL, -NOOVERFLO, -NOOVERFLOW. Default: -No_OVerFlow. Default: -No_OVerFlow.

-PRODuction; -No_PRODuction: New: -NO_PRODUCTION, -NPROD. Generates code for partial debugger functionality. Obsolete: -P, -PR, -PRO, -PRODU, -PRODUC, -PRODUCT, -PRODUCTI, -PRODUCTIO, -NOP, -NOPR, -NOPRO, -NOPROD, -NOPRODU, -NOPRODUC, -NOPRODUCT, -NOPRODUCTI, -NOPRODUCTIO, -NOPRODUCTION. Default: -No_PRODuction.

-RAnge; -No_RAnge: New: -RA, -NO_RANGE, -NRA. Generate runtime code that checks subscript ranges. Obsolete: -R, -RAN, -RANG, -NOR, -NORA, -NORAN, -NORANG, -NORANGE. Default: -No_RAnge.

-SIlent [dec]: New with decimal argument. The decimal argument is the severity level such that errors of that severity and less are not reported. If no value is given, a value of 1 is assumed. Note that a space is required between the option and the value. Obsolete: -SIL, -SIL1, -SIL2, -SIL3, -SILE, -SILEN, -SILENT, -SILENT1, -SILENT2, -SILENT3. Default: -SIlent 0

-Source pathname: This is an alternate way of specifying the source of the input file. If pathname is 'TTY', input comes from the user terminal. Obsolete: -SO, -SOU, -SOUR, -SOURC.

-SPACE: New. This option specifies that space is to be given preference over runtime speed in optimization consideration. The opposite of -SPACE is -TIME, which means that optimization is to favor runtime speed over space. Default: -TIME (that is, -SPACE is disabled).

-STATistics; -No_STATistics: New: -NO_STATISTICS. Displays compilation statistics at the terminal. Obsolete: -STATI, -STATIS, -STATIST, -STATISTI, -STATISTIC, -NOST, -NOSTA, -NOSTAT, -NOSTATI, -NOSTATIS, -NOSTATIST, -NOSTATISTI, -NOSTATISTIC, -NOSTATISTICS, -TO, -TOT, -TOTA, -TOTAL, -TOTALS. Default: -No_STATistics.

-Store_Owner_Field; -No_Store_Owner_Field: New. Causes the identity of the current routine to be stored in a known place for use by trace back routines. This obsoletes the -NO_OWNERID and -NOOWNERID options of CBL. Obsolete: -STO, -STOR, -STORE, -STORE_, -STORE_O, -STORE_OW, -STORE_OWN, -STORE_OWNE, -STORE_OWNER, -STORE_OWNER_, -STORE_OWNER_F, -STORE_OWNER_FI, -STORE_OWNER_FIE, -STORE_OWNER_FIEL, -NO_STO, -NO_STOR, -NO_STORE, -NO_STORE_, -NO_STORE_O, -NO_STORE_OW, -NO_STORE_OWN, -NO_STORE_OWNE, -NO_STORE_OWNER, -NO_STORE_OWNER_, -NO_STORE_OWNER_F, -NO_STORE_OWNER_FI, -NO_STORE_OWNER_FIE, -NO_STORE_OWNER_FIEL, -NSO, -NO_OW, -NO_OWN, -NO_OWNE, -NO_OWNER, -NO_OWNERI, -NO_OWNERID. Default: -Store_Owner_Field.

-TIME: New. Means optimization is to favor runtime speed over space. The opposite of -TIME is -SPACE, which specifies that space is to be given preference over runtime speed in optimization selection. Default: -TIME enabled.

-UPcase: Map source program to uppercase (except for quoted literals). Obsolete: -U, -UPC, -UPCA, -UPCAS. Default: -UPcase.

-XRef; -No_XRef: Produce listing with cross reference of data/procedure names. Obsolete: -X, -XRE, -NOX, -NOXRE, -NOXREF. Default: -No_XRef.

SOFTWARE PROBLEMS FIXED

A procedure's argument declared as data type (*) bit(*) now works correctly when the passed subscript value is larger than 64k (SPAR 3000678).

FORMAT statement with label in the beginning now works correctly whether it is backward or forward referenced (SPAR 3000605).

When a procedure's argument is declared as data type of bit array aligned it works correctly.

Multiple concatenations involving aggregate members and character scalars now work correctly (SPAR 3007492).

Successive function calls caused floating point inconsistencies in nested invokations. This has been corrected (SPAR 3007975).

An aggregate, no matter how complex, can now be assigned to an array member of equally declared aggregates (SPAR 3006663).

ENVIRONMENT

Standard.

Note

The EPF library COMMON_ENVELOPE.RUN is necessary to run PL1G.

INSTALLATION AND BUILD PROCEDURES

PL1G is now supplied with two directories: PL1G and PL1G_LIBRARY. In order to use PL1G, the user must run two install files, PL1G>PL1G.INSTALL.COMI and PL1G_LIBRARY>PL1G_LIBRARY.INSTALL.COMI.

The PL1G.INSTALL.COMI file now uses the PRIMOS COPY command instead of FUTIL. Since PL1G is now an EPF compiler, CMDNCO>PL1G.SAVE and the following shared segment files from SYSTEM should be deleted:

PG2044 PG2045 PG2050 PG2051 PG2216 PG2217

Also the files SYSTEM>PL1G.SHARE.COMI and TOOLS>PL1GDF.SAVE should be deleted and the line in the system start-up file PRIMOS.COMI (or C_PRMO) file that invokes the share of PL1G should be deleted.

The directory PL1G>SYSTEM has also been deleted.

All else is standard.

PMA

NEW FEATURES AND CHANGES

The opcode for SSSN is added to the opcode tables.

SOFTAWRE PROBLEMS FIXED

If the user set his own link base instead of the default in the ECB then he got an erroneous error message. This was regression at 19.2 (SPAR 3006919).

Use of the -ROUND option caused B register to be set. This could have caused erroneous conditional assemblies (SPAR 3007018).

External reference for common and external entries could have caused PMA to leave files open.

External reference for entries and DYNTS sometimes caused PMA to leave all of its files open.

OUTSTANDING PROBLEMS

A regression occurred between Rev. 19.1 and Rev. 19.2 in that R mode no longer recognizes @+ as a stack manipulation expression (SPAR 3001077).

PERMANENT RESTRICTIONS

A comment line after a PMA END statement generates error and causes SEG to reset the top of the procedure segment to 177777. This is not really a bug but is the result of the way PMA handles its input files (POLER 46341).

RPG

SOFTWARE PROBLEMS FIXED

POLER 56265 has been corrected. The execution of a particular RPG program resulted in an incorrect runtime error during the opening of files. The problem was occuring when a non-disk file was defined on a file specification form following a MIDAS file.

INSTALLATION AND BUILD PROCEDURES

Standard.

Note

This is the last time that the RPG product will be supplied to customers. Customers with maintenance agreements for RPG are entitled to receive the VRPG product and should already have been informed of this upgrade. Contact your Prime analyst or salesman for more information.

VRPG

NEW FEATURES AND CHANGES

Several new features and changes have been made to VRPG at this Revision. Many correspond to features in RPG II of IBM System/34, making VRPG more compatible with System/34 RPG. The new functionality is described in detail below. In summary, the System/34 features include:

- The specification of output fields without an end position
- An implied SR within subroutines
- Arithmetic statements with no factor 1
- Programs with no primary files
- Programs with multiple primary files
- Output only programs
- Deletion of indexed file records
- Duplicate naming allowed for a files, subroutines, and labels
- *ZERO and *BLANK
- /COPY and %INCLUDE capability
- DATE runtime option
- TREENAME runtime option
- SPOOL runtime option with -ALIAS, -AT, -FORM, and -COPIES
- Larger local data area (1024 characters)
- Specification of a program name in the Header Specification

At Rev. 19.4, a number of new compile-time options have been added to the VRPG compiler. These include the following:

-MAP	-NO_MAP
-OFFSET	-NO_OFFSET
-PRODUCTION	-NO_PRODUCTION
-RANGE	-NO_RANGE
-UPCASE	-LCASE
-ALLOW_PRECONNECTION	-NO_ALLOW_PRECONNECTION
-OPTIMIZE [n]	-FULL_OPTIMIZE
-SPACE	-TIME

-STORE_OWNER_FIELD	-NO_STORE_OWNER_FIELD
-NO_STATISTICS	
-HELP	-FULL_HELP
-NO_BINARY	-NO_LISTING
-SILENT (with a decimal argument)	

Detailed descriptions of each of these options are included here.

A number of abbreviations for compiler options have been modified. The old abbreviations will still be accepted at this revision but will not be accepted in the future.

Formation of the negated forms of those options that have them (such as -DEBUG and -NODEBUG) is now done by prefixing the name of the positive full specification with -NO_ (-DEBUG and -NO_DEBUG). To form the negation of an abbreviated option, prefix the positive abbreviated name with -N (in this case, -DBG and -NDBG). Note that the option forms -LISTING NO and -BINARY NO will still be retained for compatibility reasons, though a note will be issued by the compiler encouraging the use of the newer forms -NO_LISTING and -NO_BINARY. Also, this is the last revision that accepts the -OBDATA and -NOOBDATA options. They are the same as the -EXPLIST and -NO_EXPLIST options, which are the more acceptable forms.

Special note must be made about listing files and the use of options that imply that a listing file be produced. To begin with, options that specify that a listing is to be produced (-LISTING, -NO_LISTING or -LISTING NO) must be distinguished from those that specify what is to go into a listing if one is produced (such as -XREF, -MAP). If an option like -XREF is specified during a compiler invocation, a listing file is produced in the absence of an explicit -LISTING option. However, if -NO_LISTING happens also to be specified at the same time as -XREF, there is a conflict. Beginning at Rev. 19.4, production of a listing file is based upon a hierarchical order:

1. If -LISTING [pathname] or -NO_LISTING appears on the compiler command line, that switch determines whether or not a listing is produced independent of the appearance of any options that specify what goes into a listing. This means that -L -EXP produces a listing, and that -NL -EXP does not.

2. If -LISTING or -NO_LISTING is not explicitly set and if the positive form of an option that controls the contents of the listing appears, a listing file is produced. Note that the sole appearance of the negative form of such an option, for example -NO_MAP, does not imply a listing, and therefore none is produced.

3. If no options relating to a listing are specified on the command line, a listing is produced only if -LISTING is set as a default at a particular installation.

Following is a list of VRPG compiler options and their official abbreviations. Default options are preceded by an asterisk (*).

Option	Abbreviation
* -64V	(none)
-ALLOW_PRECONNECTION	-APRE
* -NO_ALLOW_PRECONNECTION	-NAPRE
-BANNER	-BAN
* -NO_BANNER	-NBAN
* -BINARY [pathname]	-B [pathname]
-NO_BINARY	-NB
-DEBUG	-DBG
* -NO_DEBUG	-NDBG
-ERRLIST	-ERRL
* -NO_ERRLIST	-NERRL
* -ERRTTY	-ERRT
-NO_ERRTTY	-NERRT
-EXPLIST	-EXP
* -NO_EXPLIST	-NOEXP
-FULL_HELP	-FH
-FULL_OPTIMIZE	-FOPT
-HELP	-H
-INPUT pathname	-I pathname
-LCASE	-LC
-LISTING [pathname]	-L [pathname]
* -NO_LISTING	-NL
* -MAP	-MA
-NO_MAP	-NMA
-OFFSET	-OFF
* -NO_OFFSET	-NOFF
* -OPTIMIZE [dec-integer]	-OPT [dec-integer]
-PRODUCTION	-PROD
* -NO_PRODUCTION	-NPROD
-RANGE	-RA
* -NO_RANGE	-NRA
-SEQCHK	-SEQ
* -NO_SEQCHK	-NSEQ
-SILENT [dec-integer]	-SI [dec-integer]
-SOURCE pathname	-S pathname
-SPACE	(none)
-STATISTICS	-STAT
* -NO_STATISTICS	-NSTAT
* -STATUS	(none)
-NO_STATUS	-NSTATUS
* -STORE_OWNER_FIELD	-SOF
-NO_STORE_OWNER_FIELD	-NSOF
* -TIME	(none)
* -UPCASE	-UP
-XREF	-XR
* -NO_XREF	-NXREF

The VRPG compiler now recognizes two default program suffixes, .VRPG and .RPG, in that order. When invoking VRPG with VRPG TEST, the compiler looks first for the file TEST.VRPG, then for TEST.RPG, and finally for TEST, to decide which program to compile.

The user will notice two differences in the way VRPG reports errors. The banner, which is printed after the number of errors, has changed slightly. Also, if there are any errors, a MAX SEVERITY IS # message appears after the number of errors indicating the highest severity error produced, either 1, 2, 3 or 4.

The VRPG library now comes in two varieties, NVRPGLB and VRPGLB. NVRPGLB is a nonshared library, which in previous revisions had been called VRPGLB. VRPGLB is a new shared EPF library.

The shared FORMS library is now automatically loaded when loading the VRPG library. Therefore, the load of any VRPG program which interfaces with the Prime Forms Management System (FORMS) does not have to load VFORMS.

For users of local data area, there are a couple of things to note. At previous revisions, the local data area was stored in segment 4030 and the user was responsible for initializing the segment before beginning. At Rev. 19.4, the local data area is dynamically allocated processed storage that remains accessible to the user while the user is logged in. The user is no longer required to initialize the storage himself. The storage is deallocated when the user reinitializes his command environment or when he logs out.

Note

Because of changes in VRPG compiler and library formats at this revision, any program that is to be reloaded with the VRPG library must first be recompiled. Any programs that used local data area at Rev. 19.3 and that are to share local data with VRPG programs compiled with Rev. 19.4 must be recompiled and reloaded with Rev. 19.4 VRPG.

At Rev. 19.4, the user has the choice of building and executing his runfile in two different ways, either using SEG or using BIND. BIND creates an executable program called an EPF with a .RUN suffix that is executed with RESUME rather than SEG. For further details, refer to BIND and EPF documentation.

New RPG-II Language Features

The following are new language features:

- Output fields may now be entered without the end position specified. The end position for that field is calculated to be the largest previously defined end position plus the current field's length. A severity 2 warning message is given.

- The statements within a subroutine, between the BEGSR and the ENDSR statements, may now be identified by SR or blanks in columns 7 and 8.

- The arithmetic opcodes ADD, SUB, MULT, and DIV may now be used without factor 1 specified. In such cases, the result field is substituted for factor 1, the operation is performed, and the resulting value is placed back in the result field.

- The Primary file, the main input file in an RPG program, is no longer required. Previously, programs were assumed to be single file processing by defining a primary file or multiple file processing by defining a primary file and secondary files. The program execution would normally end when one or all of these files had all records processed. If the primary file is not used, the program should provide another means of exit by turning on the last record indicator (LR) within the calculations. If a primary file is not specified in a program and one or more secondary files are specified, the first secondary file is processed as the primary file.

- The specification of multiple primary files within a program is now also allowed. It is assumed that external indicators are used to determine which primary file is to be used during the program execution. If an attempt is made to use more than one primary file during execution, one is treated as the primary file and the other files are assigned as secondary files. If no primary file is used during execution and secondary files are used, the first secondary file is treated as the primary file.

- It is now possible to write output-only programs. Note that the program should provide some means of exiting, preferably by turning on the LR indicator within the calculation section.

- VRPG now supports deletion of records for indexed files. In order to delete records from a file, the file must be specified as an update file. Deletion is specified in the output specifications by entering DEL in columns 16-18 of the main line in an output description, along with the filename, the type of output, and any output indicators.

  The method for deleting records is similar to the method for updating records. The deletion must be preceded by the retrieval of a record. It is this record that is deleted. A chained file or a demand file can be deleted at detail time, at

total time, or at exception time. However, all other disk files should be deleted only at detail or exception output time during the same program cycle in which the record is read. A runtime error is given if the program attempts to delete a record when there was no successful preceding read. The user has the option of continuing the execution and ignoring that particular deletion attempt.

OR lines may be used to condition the deletion but DEL should only be specified in the first line of the specification. DEL applies to all of the OR lines. No field specifications should given for a deletion.

- Filenames are no longer required to be unique names. A filename may have the same name as a field, array, table, subroutine, or label used within a program. For program clarity, it is recommended that unique names be used.

  Subroutines and labels are no longer required to have unique names. An internal subroutine (defined with BEGSR) or a label may have the same name as a file, field, array, or table used within a program. If the Source Level Debugger, DBG, is used when a label or subroutine name has the same name as a field, table, or array, the label or subroutine should be referenced by its name prefixed with an ampersand (&). A compiler warning message appears if a non-unique name is used and the program is compiled with the -DEBUG option. For program clarity and ease of debugging, it is recommended that unique names be used.

- Four new figurative constants are now allowed in RPG programs, *ZERO, *ZEROS, *BLANK, and *BLANKS. These may be used only on calculation statements as either factor 1 or factor 2. They may not be used with move zone operations, bit operations, or the DEBUG, DSPLY, or SQRT opcodes. *ZERO/*ZEROS is a field of all zeros and may be used with numeric or alphanumeric fields. *BLANK/*BLANKS is a field of all blanks, and may be used only with alphanumeric fields. The length of the figurative constant is assumed to be equal to the length of the other factor field, if present. Otherwise, it is assumed to be equal to the length of the result field. Figurative constants are considered to be elementary items and, if used with an array, act like a field. For example, MOVE *ZEROS ARR1 would fill the entire array with zeros.

- /COPY or %INCLUDE file capability is now available with VRPG. Enter /COPY or %INCLUDE starting in column 7 followed by a space and the filename or pathname of the file to be included in the source code. INCLUDE files may be used anywhere in the program and may themselves contain INCLUDE files. The included text can be seen in the listing file.

- A new runtime option, -DATE or -D, has been added to allow the user to set the system date. The date wanted must be entered following -DATE in the format MM/DD/YY. Any other format will

cause a warning message and the current date will be substituted for the one given.

- An alternate way, -TREENAME or -T, is now available for requesting runtime file assignments. This may be used instead of using T in column 52 of the header specifcation.

- Printer output files may now be sent directly to the spooler by the use of the -SPOOL runtime option. This option can optionally be used with any or all of the following runtime options:

  -ALIAS <alias-name>-AT <destination>,
  -FORM <form-type>-COPIES <number>

  Accepted abbreviations are -S for -SPOOL, -AS for -ALIAS, -A for -AT, -F for -FORM, and -C for -COPIES. If -SPOOL is used, all output printer files are spooled and they are not written on disk.

- A user's local data area now has a maximum size of 1024 characters.

- A user may now assign a name to his program in columns 75 through 80 of the header specification. If none is assigned or if there is no header specification, the program name defaults to RPG$MAIN. This name must be unique in your program. The program name now appears on the cross reference listing of your program even if none is assigned. The source level debugger, DBG, uses the program name as the main program block of an RPG program.

New Compiler Options

-MAP/-NO_MAP Options: The -MAP option produces information similar to the -XREF option but without the source line references. See the VRPG Reference Guide (IDR5040) for an example of output produced by the -XREF option. When you specify -MAP, you automatically have a listing file produced. Conversely, when you specify -L, you automatically get the map information. If you do not want your listing file to have the map information, you need to specify -L with -NO_MAP.

-OFFSET/-NO_OFFSET Options: The -OFFSET option appends an offset map to the end of the listing file. The use of -OFFSET implies -L. For each statement in the source program, the offset map gives the offset in the object file of the first machine instruction generated for that statement. The -NO_OFFSET option means that no offset map is created and is the default.

-PRODUCTION/-NO_PRODUCTION Options: These options are alternative options controlling code for Prime's Source Level Debugger, DBG. The -PRODUCTION option is similar to -DEBUG except that no statement information is produced. You may not breakpoint at individual statements nor step through your program with -PRODUCTION. The execution time of a program compiled with -PRODUCTION is better than the execution time of a program compiled with -DEBUG. The -NO_PRODUCTION option is the default.

-RANGE/-NO_RANGE Options: These compiler options control runtime subscript range checking. When -RANGE is specified on the compile line, any variable array subscript is checked for out-of-range conditions and, if any occurs, a runtime error is issued and the program halts. The use of -RANGE increases both the compile time and the execution time of a program and therefore should be used only as a debugging tool. The -NO_RANGE option suppresses this capability and is the default option.

-UPCASE/-LCASE Options: These compiler options control the compiler's automatic conversion to upper case. When -UPCASE is used, everything in the source program, except literals, is translated to upper case. When -LCASE is used, the source program (except for reserved words) is left as is. A program can therefore have two data items, 'X' and 'x' for example, which are treated as separate items with -LCASE. The -UPCASE option is the default.

-ALLOW_PRECONNECTION/-NO_ALLOW_PRECONNECTION Options: If this option is specified, preconnection of the listing output to a pre-opened file unit 2 or of the binary output to a pre-opened file unit 3 is allowed. If the negative form is specified, the compiler always opens and closes the listing and the binary files (and uses dynamic file units). The negative form is the default.

-OPTIMIZE [n] Option: The VRPG compiler now allows for optimization of object code. Optimization may be specified at one of three different levels indicated by the decimal number n, which can be either 1, 2, or 3. Optimization level 1 does code pattern replacement. Optimization level 2 performs code pattern replacement and redundancy elimination. Optimization level 3 does code pattern replacement, redundancy elimination, and also takes invariant code out of loops. If the decimal integer is not specified, a default value of 2 is used. If no optimization is desired. a value of 0 should be used. It should be remembered that optimized code runs more efficiently than non-optimized code but that it takes somewhat longer to compile.

-FULL_OPTIMIZE Option: This option indicates that the highest possible optimization should be used and is equivalent to -OPTIMIZE 3.

-SPACE and -TIME Options: The -SPACE option specifies that the size of the optimized code (space) is to be given preference over the speed of the optimized code (time) in optimization consideration. The opposite of -SPACE is -TIME, which means that optimization is to favor time over space. The -TIME option is the default.

-STORE_OWNER_FIELD/-NO_STORE_OWNER_FIELD Options: The -STORE_OWNER_FIELD option causes the identity of the current program to be stored in a known place for use by trace back routines such as DMSTK. This option is the default. Using the -NO_STORE_OWNER_FIELD option does not save this information.

-NO_STATISTICS Option: This compiler option suppresses compile time statistics, which may be generated by the -STATISTICS option. See the VRPG Reference Guide (IDR5040) for information on the statistics generated by the -STATISTICS option. The -NO_STATISTICS option is the default.

-HELP Option: This new option produces information on using the compiler, including a list of all compiler options. When this option is used, the help information only is issued and the compiler is not invoked.

-FULL_HELP Option: This new option is similar to the -HELP option but, in addition to the usage summary, a description of the meaning of each compiler option is also given.

-NO_BINARY Option: This option takes the place of using -BINARY NO, which will soon be obsolete, to specify that no binary file is to be produced.

-NO_LISTING Option: This option takes the place of using -LISTING NO, which will soon be obsolete, to specify that no listing file is to be produced. The default is to not produce a listing file.

-SILENT [n] Option: This option has been modified to accept a decimal argument, which indicates an error severity level. When n is specified, all errors with that severity and less are not reported. If n is omitted, a severity level of 1 is assumed, as in the past. There must be a space between the option and the value. A level of -1 means that -SILENT was not specified and that all error messages will appear. This is the default.

SOFTWARE PROBLEMS FIXED

RPG programs generated extra code, which was a problem with some very large programs. At this revision of VRPG, most unused information is no longer written into the binary file (SPAR 3001747).

The source level debugger, DBG, was unable to evaluate the table index for a table having a name less than 6 characters long. This has been corrected (SPAR 3002283).

A problem with output of a data structure subfield using an edit word has been fixed and now works correctly (SPAR 3003556).

An illegal entry in factor 2 of an EXIT statement is now reported as an error (SPAR 3004236).

An overflow indicator is now properly set in a program in which an overflow indicator is assigned but not used to condition output (SPAR 3004320).

The use of conditioning indicators in calculation and output statements is now properly handled. Prior to Rev. 19.4, if an indicator in any of these positions was not previously defined in the program, the indicator was generally off and no calculation or output occurred. The indicators 01-99, L1-L9, H1-H9, OA-OV, and KA-KY should be defined elsewhere in the program to be used as conditioning indicators. If they are not, a severity 1 warning is now issued (SPAR 3004395).

VRPG programs can now have two primary files (SPAR 2004544).

More incorrect entry errors are now reported. For example, a severity 3 error message reports the use of a subroutine name on an output field. Similarly, an error is reported if an array subscript is not a numeric field or a literal constant. Also, if a semantic error is reported on a field within an input or an output record, the rest of the fields within the input or the output record are examined. For some cases in the past, no other semantic errors were reported for the following fields in the record (SPAR 3004805).

A data structure subfield is now accessible for evaluation or modification during a DBG session. Also, if a program does not specify a data structure name, one is assigned by the compiler in the form of ds_1, ds_2, and so on in the order the structures are defined in the program (SPAR 3004806).

VRPG now follows the standard Prime suffix convention for filenames. To be compatible with previous revisions, VRPG recognizes and accepts either the .VRPG or .RPG suffix, in that order (SPAR 3006985).

If a lookahead record was defined before other input record types, usually numeric sequence types, reading from the file caused the status of the record id indicators to be invalid and subsequent calculations and output conditioned by those indicators never occurred. This has been corrected (SPAR 3007426).

OUTSTANDING PROBLEMS

It is not possible to set a break point on all TAG statements in VRPG programs. A temporary workaround is to set the break point on the next executable calculation statement (SPAR 3006403).

ENVIRONMENT

The VRPG compiler needs the file COMMON_ENVELOPE.RUN in the LIBRARIES* directory to run.

INSTALLATION AND BUILD PROCEDURES

VRPG is now supplied in two directories, VRPG and VRPG_LIBRARY. In order to use VRPG, the user must run two install files, VRPG.INSTALL.COMI from the UFD VRPG and VRPG_LIBRARY.INSTALL.COMI from the UFD VRPG_LIBRARY.

The default options of the VRPG compiler may be changed by your System Administrator by running TOOLS>RPGDF.CPL. When this is done, the name of the error file to be used is now called VRPGDATA instead of RPGDATA. See the System Administrator's Guide for further information on changing compiler options.

Note that the following files are no longer needed and should be deleted by your System Administrator:

CMDNCO>VRPG.SAVE	SYSTEM>VRPG.SHARE.COMI,
SYSTEM>VR2142	SYSTEM>VR2143
SYSTEM>VR2144	SYSTEM>VR2145
SYSTEM>VR2146	TOOLS>RPGDF.SAVE

The line sharing the compiler in the system start-up file PRIMOS.COMI should also be deleted.

CHAPTER 5

DATA MANAGEMENT SYSTEMS

DBMS

NEW FEATURES AND CHANGES

F77 Interface

This release provides an interface between DBMS and the F77 compiler. If you have F77 installed on your system and plan to use it for writing DBMS DML programs, you must obtain the UFD DBMSF77 from the system tape. DBMSF77 has a structure identical to that of DBMSFTN; see FILES ON SYSTEM TAPE of this document.

The utilities are invoked in the same manner for DBMS F77 subschemas and DML applications as they are for DBMS FTN subschemas and applications. See CREATION OF A DML APPLICATION PROGRAM for a description of how to create F77 DML application programs.

Features of F77 include support for long variable names, the CHARACTER data type, LOGICAL*4 default for logical data type declaration, and INTEGER*4 default for integer data type declaration. F77SUBS (F77 subshema compiler) and F77DML (F77 preprocessor) support these features in the following manner (note the distinctions between F77 and the DBMS support of F77):

- Data names may be up to 30 characters in length (versus F77's 32 character name length support) and may include the $ (dollar sign) and _ (underscore) characters.

- Character strings are supported with a new syntax; for example, CHARACTER*14 FIRST_NAME as opposed to FTN's CHARACTER FIRST_NAME = FNAME(7).

Note

Character string length must be declared as an even number.

- Variables declared as INTEGER or LOGICAL in an F77 subschema will be stored as half-words; that is, 16 bits versus F77's default to full 32-bit words, and variables declared as INTEGER*4 in an F77 subschema will be stored as full words (32 bits).

DMLCP

Two new formats of the FIND/FETCH DML statements have been added to support partial key processing:

Format 7 — Partial Key Search:

{ FETCH / FIND } record VIA [CURRENT OF] SET set SUBKEYED item-list.

This statement bases selection on the values contained in item-list. The items in item-list are implicitly qualified by record and must constitute a leftmost subset of a search or sort list in set containing record. No non-key items are allowed. DBMS then selects a record having values for items in the item-list that are greater than or equal to the values in the User Work Area. If more than one record occurrence meets these criteria, DBMS selects the first record it finds. (The set is searched in the NEXT direction.)

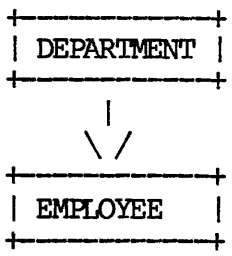
If the CURRENT phrase is used, DBMS bases record selection on the current set occurrence of set. If the CURRENT phrase is not used, DBMS bases record selection on the SET OCCURRENCE SELECTION clause defined for the named record and set.

Though this format is similar to Format 5, some important differences should be noted:

- The statement will not fail if no equality match is made. It only fails with a 'No Record Satisfies Find' exception if the values in the User Work Area are beyond all entries in the search list for the designated set occurrence.

- The SUBKEYED clause is required in all cases. The items in item-list must constitute a leftmost subset of a search or sort key in set. No non-key items are allowed.

For example, assume the following set structure having a SEARCH list with concatenated key items HIRE-DATE, LAST-NAME:




```
#FETCH EMPLOYEE VIA CURRENT OF SET DEPT-EMP SUBKEYED HIRE-DATE.
```

Assume the HIRE-DATE field in the User Work Area was assigned the value '06/01/83'. The above DML statement will retrieve the employee record for the first employee in the current department who was hired on or after June 1, 1983. If no employees have been hired in that department since May of 1983, the statement will return a 'No Record Satisfies Find' exception -- 2226.

Format 8 -- Relative Access of Search or Sort List:

```
{ FETCH } { NEXT  }
{ FIND  } { PRIOR } RECORD [record] OF CURRENT LIST OF SET set.
```

This format is designed to take advantage of the sort order inherent in search lists. This functionality has previously been available only for sort lists (member lists) through Format 3. It requires the introduction of the concept of the "current list".

Whenever a record is accessed through its set (Formats 5, 6 or 7, and some variations of Formats 2 and 3), the search or sort list that was used to locate that record becomes the current list of that set. In most cases, that list is the member list. But Formats 5, 6 and 7 allow access through a search list when the items in the USING clause or the SUBKEYED clause are a leftmost subset of a search key. In such cases, the current list is a search list. Format 8 may be used to access that search list in either a sequential or reverse order.

This format requires a current member, a current owner, and a current list. The current list may be either a sort list (member list) or a search list. If record is specified, that record type must be

contained in the current list at runtime. It should be noted that, unlike Format 3, a FIND NEXT or FIND PRIOR where no current member exists results in an exception. It does not become equivalent to a FIND FIRST or FIND LAST, respectively.

The most common use of this format will be within a loop following a Format 7. Where the FIND NEXT variation is used, it will result in the retrieval of records greater than or equal to the values in the SUBKEYED item-list. Where the FIND PRIOR variation is used, it will result in the retrieval of records less than (but not necessarily equal to) the values in the SUBKEYED item-list. Special care must be taken in programming with the FIND PRIOR variation. Format 7 retrieves the first record greater than or equal to the item-list. Additionally, a subsequent FIND PRIOR will not retrieve any records containing keys which are duplicates of the values in the User Work Area. Therefore the item-list should be initialized in the User Work Area to a value just beyond the upper bound of the key values desired. Then FIND PRIOR may be used to effect a strictly "less than" retrieval.

Continuing the example of Format 7, if it were desired to know the names of all employees in the department hired on or after 6/1/83, the statement could be followed with:

```
10    #FETCH NEXT RECORD OF CURRENT LIST OF SET DEPT-EMP.
C
C      Print Employee Name...
C
      IF (ERSTAT.EQ.0) GO TO 10
```

The current list would be the search list containing the key items HIRE-DATE, LAST-NAME. The above statement would retrieve employee records in the order in which the employees were hired.

New Exception Codes: The following are new exception codes with explanations:

0324 — RECORD TYPE NOT INCLUDED IN CURRENT SET LIST

The above exception may occur in Format 8 when the record type is specified and the set is a multi-member set. Member lists may contain more than one record type but search lists always contain just one. If the list that is current is a search or member list that does not contain the specified record type, the exception will occur.

0328 — NO CURRENT LIST OF NAMED SET

A current list is established whenever a record is accessed through the set. Retrieval of a record through its DBK (Format 1), its area file (Formats 2 and 3), or its calc key (Format 4) are examples of record retrieval without using a set. If no current list has been established prior to a FIND Format 8, this exception occurs.

MODIFY Performance Optimization

A rewrite and optimization of the MODIFY command is estimated to result in an average of 20 to 50% runtime improvement for MODIFY processing. In addition to general MODIFY processing optimization, the algorithm has been changed to check whether an item in a key has changed before removing it from any set lists or deleting it from the calc file.

SCHEMA Compiler

The SCHEMA compiler now supports schemas of up to 65534 half-words. Schema size was previously restricted to 32767 half-words. DMLCP and all DBMS subproducts now support the increased schema table size.

Mixed Mode Transactions

At Rev. 19.4, DBMS, ROAM, and PRISAM support mixed mode transactions. This new functionality allows you to access PRISAM and DBMS files from within the same transaction from a DBMS or PRISAM application program (SPAR 3006498).

Using Mixed Mode Transactions: DBMS applications using mixed mode transactions can be written in any DBMS data manipulation language (COBOL, CBL, FORTRAN, or F77 DML). To use mixed mode transactions, you must invoke both DBMS and PRISAM prior to the start of the transaction. This signals ROAM to expect the mixed mode transaction.

In a mixed mode transaction, you may issue the calls to START, END, or ABORT the transaction through either DBMS or PRISAM. However, the user-supplied transaction identifier or number for START, END, and ABORT must be the same regardless of which subsystem you call.

Both DBMS and PRISAM 'START TRANSACTION' calls specify whether the transaction will perform an update or a retrieval. The update or retrieval parameter applies to all DBMS and PRISAM files used in the transaction.

Loading the PRISAM Library: When you load the DML program that contains mixed mode transactions, you must use the -PRISAM or -PSM option on the command line to load the PRISAM library, as well as the

DBMS library, with the application. This option applies to loads for COBOL, CBL, FORTRAN, and F77. For example, if you use the CPL procedure to compile and load a CBL DML program, the command line should be as follows:

```
R DBMSLB>CBLDML program-name -PRISAM
```

The example below shows a portion of a COBOL DML program that starts the transaction through DBMS, accesses both a PRISAM file and a DBMS file, and ends the transaction through PRISAM. Each statement is commented.

```
#INVOKE DBMS.                                  /* DBMS invoke
OPEN I-O MMT                                   /* open PRISAM file
#OPEN ALL AREAS                                /* open DBMS areas
#START TRANSACTION TRAN-ID.                    /* start DBMS transaction
READ MMT RECORD KEY IS MMT-KEY-2               /* read from PRISAM file
#FETCH RECORD A1R1-REC.                        /* fetch DBMS record
CALL 'Z$ENDT' USING TRAN-ID, STATUS-CODE. /* end trans. through PRISAM
#CLOSE ALL AREAS                               /* close DBMS areas
#EXIT DBMS                                     /* exit DBMS
CLOSE MMT                                      /* close PRISAM
```

Mixed Mode Currency Handling: PRISAM START and END transaction calls can specify whether currency will be cleared for the PRISAM file. This has no effect on the currency of the DBMS file. A DBMS 'START TRANSACTION' implies that currency for the DBMS subsystem is the same as before the transaction starts. In the case of a PRIMEWAY user of the DBMS subsystem, however, currency is cleared, resulting in no current records at the start of the transaction.

Mixed Mode Error Handling: Mixed mode errors are reported in the usual manner to the application program by the subsystem in which they occur: to the ERSTAT status register for DBMS and as a return code for PRISAM. If the error causes the transaction to abort, the transaction is aborted for both subsystems. If the error causes an exit and cleanup to occur, the exit and cleanup occurs in both subsystems.

For example, the following errors could be returned by a DBMS START or ABORT TRANSACTION statement only when a failure occurs in the PRISAM START or ABORT transaction procedures:

ERSTAT	CONTYP	Description
38F	0	Fatal error in non-DBMS subsystem
35	28	Transaction aborted by non-DBMS subsystem
76	0	Error in non-DBMS subsystem

In addition, if a PRISAM OPEN or START transaction call or a DBMS INVOKE statement occurs within a transaction, the following error occurs:

ERSTAT	CONTYP	Description
16F	29	Mixed mode transaction already active

A transaction identifier mismatch between the END or ABORT transaction and the START transaction is handled differently by PRISAM and DBMS. If the END or ABORT transaction is a DBMS call, the identifier mismatch causes the transaction to abort. If the END or ABORT transaction is a PRISAM call, the identifier mismatch generates a PRISAM error return but does not abort the transaction. It is up to the application to trap this error and perform appropriate action.

More information about the PRISAM errors is recorded in the PRISAM error log. (See the PRISAM User's Guide.)

Schema Subfiles

DBMS can now handle schemas that define as many as 254 area, calc, and set subfiles (SPAR 3003517).

DBACP

VERIFY SUBSCHEMA no longer shows User Work Area size as negative when User Work Area is greater than 32K bytes (FORTRAN subschemas only).

VERIFY SUBSCHEMA now correctly displays a subschema's language type for all languages (SPAR 3007797).

EXPAND no longer clears the date-time saved stamp if the user answers NO to an ARE YOU SURE? prompt when after-imaging is enabled for the schema.

The screen display for EXAMINE SET and PACK SET now scrolls correctly.

DMLCP

Concurrency code 20 is documented as 'Reserved for Future Use' in the DBMS Data Manipulation Language Reference Guide. This code is currently used and is documented in the Rev. 19.4 update as:

START, END/ABORT TRANSACTION IDS DO NOT MATCH

This error is a DML program error.

FIND NEXT DUPLICATE on calc records no longer loses the remaining records on the calc chain due to infinite looping.

DBMS now correctly converts any blanks in a numeric picture item in a calc key to zeros prior to hashing the calc file key on a #MODIFY.

DMLCP no longer aborts commands because it has run out of memory to support the command rewind functionality. Previously, DMLCP made use of a 16K word array in memory to log file modifications within any one DML command. If these modifications overflowed the array (as might happen in the case of a #DELETE MANDATORY), a fatal error was generated. This functionality has been modified so that if buffer overflow occurs a temporary file (T$nnnn) containing the contents of the buffer is written to disk in the DBMSLB directory. This file is created only on memory buffer overflow and is deleted at the completion (successful or not) of the command. It is appended as many times as is necessary to complete the command (SPAR 3000714).

DMLCP now correctly checks privacy keys from DISCOVER (SPAR 3003637).

DMLCP no longer has code to disable sharing area file headers (SPAR 3003640).

Member records whose owner record is the system or is in a different area are now clustered (SPAR 3004319).

DBMS now allows access to DBMS and PRISAM files from within the same transaction (SPAR 3006498).

When executing a DML MODIFY command without an item list, DMLCP now correctly retains all information for items that are part of repeating groups not included in the subschema (SPAR 3006601).

DMLCP now correctly maintains currency when a DML 'FIND CURRENT OF SET' is executed. Previously, the current record of run-unit was occasionally not updated on executing this statement (SPAR 3007232).

The DML 'IF' command now correctly indicates that a record occurrence is not a member of any set if it has in fact not been inserted into any set (SPAR 3007757).

When executing a DML MODIFY command, DMLCP now correctly reclaims space from obsolete record fragments that are at the bottom of dropped buckets (SPAR 3008045).

CSUBS/FSUBS

CSUBS and FSUBS now correctly handle a subschema containing a 30 character data name. Previously the compilation aborted with a 'Stringrange' error.

COBOL subschema listings now show the last byte field with the correct length.

DBUTL

Issuing a DBUTL VER command with no current set now reports that there is no current set and returns to DBUTL command level (SPAR 3003002).

The DBUTL ROAM command now warns the user if an attempt is made to convert a file on a remote machine and leaves the file as it originally was (SPAR 3003111).

The DBUTL ROAM command now deletes extraneous subfiles encountered while converting RAM files to ROAM files. The volume and subfile number are displayed for every subfile deleted. These extraneous subfiles are subfiles that cannot be verified via DBACP because DBMS has no record of them. Such subfiles may have been inadvertently created by DBMS/RAM. The importance of saving the RAM schema to tape before doing the conversion cannot be emphasized too strongly; any subfile that DBMS does not know about will be deleted and cannot otherwise be restored (SPAR 3003570).

DBUTL's REWIND command now correctly rewinds a record, set, or area after a current of file type has been selected via the RDIR, SDIR, or ADIR command, respectively. This command has worked and still works for a set or area when the current of file type is selected via the SET or AREA command, respectively (SPAR 3003644).

The attempted conversion of a ROAM schema via DBUTL's ROAM command no longer results in the 'File Not Found' message (SPAR 3003957).

DBUTL's NODE command now displays more than 79 characters (SPAR 3006039).

DBUTL's RAM command is now much more robust. The user is given the option within the RAM conversion of generating a COMOUTPUT file of the conversion session. This is strongly recommended if the user has not already started one before entering DBUTL or has not already used DBUTL's OUTPUT command. A list of restrictions for this command are displayed to the screen before any conversion is done. There is much

more error checking to verify that the environment is satisfactory for the conversion. A schema is no longer renamed and renumbered unless there is already an entry for the schema in the RAM SCHDIR. Segment directories are now copied automatically to specified PDBMS UFDs. Breaks are disabled during the actual conversion of the file. Subschemas are now properly converted (SPAR 3006126).

Other Changes

SCHED now correctly sets up list headers and owner directories when adding a new set (SPAR 3009351).

DMLCP now allows you to remove a member from a set from which currency has been suppressed (SPAR 3000133).

CSUBS now aligns a repeating group, which is not word-aligned, as it did before Rev. 19.1. This means CSUBS no longer pads the group to word alignment as CBLSUBS does (SPAR 3002634).

CSUBS assigns correct default size of PIC(6) to a level 77 item if the item type is DBK (SPAR 3003296).

The RECFLG and ERITEM variables are now declared consistently (SPAR 3004215).

A third argument was added to the calling sequence of the OS routine RINSG$. Prior to Rev. 19.3, this argument is not required (SPAR 3004306).

Two buffers per user have been added to DMLCP. These buffers act as windows into data base files and allow DMLCP to reduce the number of calls to ROAM, especially in read operations (SPAR 3006239).

DBMS no longer generates an internal fatal error when an attempt is made to store more than 256 entries in one bucket (SPAR 3007575).

DBMS now processes signals in the same manner as ROAM (SPAR 3007423).

FSUBS does not check the use of reserved words for Schema name (SPAR 3002581).

ENVIRONMENT

Rev. 19.4 DBMS requires PRIMOS Rev. 19.4, ROAM Rev. 19.4, and SEG Rev. 19.0 or greater. DBMS runtime (DMLCP) requires the use of shared segments 2001, 2002, 2003, and 2012 as well as private segments 4030, 4031, 4032, 4033, and 4034.

INSTALLATION AND BUILD PROCEDURES

The DBMS build procedure is standard. A description of DBMS installation procedures follows.

Files on System Tape

DBMSEX (UFD): This is the primary packaged UFD supplied on the system tape. It is a prerequisite for all the others. By itself it is adequate to allow for execution only of DBMS. Its UFD structure is followed by each of the other packaged UFDs. Exceptions will be noted where appropriate.

The three files at the top level are the install procedures DBMS.INSTALL.CPL and DBMS.INSTALL.COMI plus the share procedure DBMS.SHARE.COMI. To meet Master Disk standards, the install file, DBMS.INSTALL.COMI, can now be invoked as a COMINPUT file. It simply executes the old install file DBMS.INSTALL.CPL, which is still of necessity a CPL routine. The install files for each of the individual DBMS packaged UFDs are also still CPL routines.

DBMSEX>DBMSLB (Sub-UFD): Here we find the shareable run files, segment directories, and support utilities that are moved to top level DBMSLB by DBMS.INSTALL:

DB2001	DB2002	DB2003	DB2012	DB4000
DB2070	VFYPRT.SAVE	DUMP.SEG	IDBMS.SEG	
DBACP.SEG	DBUTL.SEG	SUMMARY.SAVE	DAEERS	DBMSE

DBMSEX>CMDNC0: This UFD holds the CPL interludes to those V-mode products found in DBMSLB that become external commands to PRIMOS. These interludes are moved to the top level CMDNC0 by DBMS.INSTALL.

DBACP.CPL DBUTL.CPL

DBMSEX>BINARY (Sub-UFD): Each sub-product in a package has a binary file in this UFD of the form %prod%.BIN. In addition, the following files are produced by LOAD_LIB:

DBDATA.BIN VERSIO.BIN LIBRARY.MAP

DBMSEX>JOBS (Sub-UFD): Here we have the procedures of the form %prod%_LOAD.CPL that produce the run files or segment directories in DBMSEX>DBMSLB for this package. In the case of DMLCP, the routine is called LOAD_LIB.CPL and it, in turn, requires some additional files:

DBDATA.FIN VERSIO.FIN HTAB.INS.FIN DYNT.PMA

Also found here are two utilities for setting up special configuration options at the time of the install and for automating reloads of subproducts. (See following sections Introductory Message Control and Reloading Subproducts.)

SETUP.CPL RESET_UFD.CPL

DBMSEX>INFO (Sub-UFD): This UFD contains the documentation for the current release of DBMS.

DBMSDEF (UFD): This UFD has a similar structure to DBMSEX. It is packaged with DBMSEX for systems on which there will be a need to develop schemas, so the sub-products contained herein are SCHEMA and SCHDEC. The install file DBMSDEF.INSTALL.CPL is here at the top level.

DBMSDEF>DBMSLB (Sub-UFD): SCHEMA.SEG SCHDEC.SEG

DBMSDEF>CMDNC0 (Sub-UFD): SCHEMA.CPL SCHDEC.CPL EXEC.SAVE

DBMSDEF>BINARY (Sub-UFD): SCHEMA.BIN SCHDEC.BIN EXEC.BIN

DBMSDEF>JOBS (Sub-UFD): SCHEMA_LOAD.CPL SCHDEC_LOAD.CPL
EXEC_LOAD.CPL

DBMSFIN (UFD): This UFD is packaged with DBMSDEF and DBMSEX for those systems on which FIN applications will be developed. Thus, the sub-products FSUBS and DML as well as the LIB entry point file DMLLIB are included. The only file at the top level is DBMSFIN.INSTALL.CPL.

DBMSFIN>DBMSLB (Sub-UFD): FSUBS.SEG DML.SEG

DBMSFIN>CMDNC0 (Sub-UFD): FSUBS.CPL FDML.CPL

DBMSFTN>LIB (Sub-UFD): DMLLIB.BIN

DBMSFTN>BINARY (Sub-UFD): FSUBS.BIN DML.BIN

DBMSFTN>JOBS (Sub-UFD): Here, in addition to FSUBS_LOAD.CPL and FDML_LOAD.CPL, are two utilities, FDML.CPL and FLOAD.CPL, for use in precompiling and loading applications programs. (See following section Creation of a DML Application Program.) They are moved to the top level UFD DBMSLB by the INSTALL procedure.

DBMSF77 (UFD): This UFD is packaged with DBMSDEF and DBMSEX for those systems on which F77 applications will be developed. Thus, the sub-products F77SUBS and DML as well as the LIB entry point file DMLLIB are included. The only file at the top level is DBMSF77.INSTALL.CPL.

DBMSF77>DBMSLB (Sub-UFD): F77SUBS.SEG DML.SEG

DBMSF77>CMDNC0 (Sub-UFD): F77SUBS.CPL F77DML.CPL

DBMSF77>LIB (Sub-UFD): DMLLIB.BIN

DBMSF77>BINARY (Sub-UFD): F77SUBS.BIN DML.BIN

DBMSF77>JOBS (Sub-UFD): Here, in addition to F77SUBS_LOAD.CPL and F77DML_LOAD.CPL, are the two utilities F77DML.CPL and F77LOAD.CPL for use in precompiling and loading applications programs. (See following section Creation of a DML Application Program.) They are moved to the top level UFD DBMSLB by the INSTALL procedure.

DBMSCOB (UFD): This UFD is packaged with DBMSDEF and DBMSEX for those systems on which COBOL applications will be developed. Thus, the sub-products CSUBS and DML as well as the LIB entry point file DMLLIB are included. The only file at the top level is DBMSCOB.INSTALL.CPL.

DBMSCOB>DBMSLB (Sub-UFD): CSUBS.SEG DML.SEG

DBMSCOB>CMDNC0 (Sub-UFD): CSUBS.CPL CDML.CPL

DBMSCOB>LIB (Sub-UFD): DMLLIB.BIN

DBMSCOB>BINARY (Sub-UFD): CSUBS.BIN DML.BIN

DBMSCOB>JOBS (Sub-UFD): Here, in addition to CSUBS_LOAD.CPL and CDML_LOAD.CPL, are the two utilities CDML.CPL and CLOAD.CPL for use in precompiling and loading applications programs. (See following section Creation of a DML Application Program.) They are moved to the top level UFD DBMSLB by the INSTALL procedure.

DBMSCBL (UFD): This UFD is packaged with DBMSDEF and DBMSEX for those systems on which the new CBL (ANSI 1974 COBOL) compiler is installed and will be used to develop applications. Thus, the sub-products CBLSUBS and DML as well as the LIB entry point file DMLLIB are included. The only file at the top level is DBMSCBL.INSTALL.CPL.

DBMSCBL>DBMSLB (Sub-UFD): CBLSUBS.SEG DML.SEG

DBMSCBL>CMDNCO (Sub-UFD): CBLSUBS.CPL CBLDML.CPL

DBMSCBL>LIB (Sub-UFD): DMLLIB.BIN

DBMSCBL>BINARY (Sub-UFD): CBLSUBS.BIN DML.BIN

DBMSCBL>JOBS (Sub-UFD): Here, in addition to CBLSUBS_LOAD.CPL and CBLDML_LOAD.CPL, are the two utilities CBLDML.CPL and CBLLOAD.CPL for use in precompiling and loading applications programs. (See following section Creation of a DML Application Program.) They are moved to the top level UFD DBMSLB by the INSTALL procedure.

DBMSLGCL (UFD): This UFD is packaged with DBMSDEF and DBMSEX for use on systems where the schema editor is needed. The install file SCHED.INSTALL.CPL is here at the top level.

DBMSLGCL>DBMSLB (Sub-UFD): SCHED.SEG

DBMSLGCL>CMDNCO (Sub-UFD): SCHED.CPL

DBMSLGCL>BINARY (Sub-UFD): SCHED.BIN TEXTED.BIN

DBMSLGCL>JOBS (Sub-UFD): SCHED_LOAD.CPL

Instructions for Initial Installation of DBMS

1. If you already have a version of DBMS on your system, see the section Upgrading an Existing DBMS Installation.

2. Restore the UFDs supplied on tape. These may be one or more of the following:

```
DBMSEX   DBMSDEF  DBMSFTN  DBMSF77
DBMSCOB  DBMSCBL  DBMSLGCL
```

3. Install and share ROAM on your system (see separate ROAM documentation).

4. Once the various UFDs described above have been created, attach to the MFD where you want DBMS to reside and issue the command:

```
CREATE DBMSLB
```

5. Run SETUP if you want your configuration options to be non-standard. (See following section Introductory Message Control.)

6. If you did not run it already as part of step 5 above, issue the command:

```
CO DBMSEX>DBMS.INSTALL.COMI
```

7. Share DBMS from the supervisor terminal thus:

```
CO SYSTEM>DBMS.SHARE.COMI
```

   Be sure to put the above share command into PRIMOS.COMI (C_PRMO) in CMDNCO so that DBMS will be shared at any time the system is cold started (see section DMLCP Installation).

8. Finally, DELETE the UFD(s) restored from tape.

Upgrading an Existing DBMS Installation

1. Save all Schemas to tape.

2. If you are upgrading from Rev. 18.2 or later, you should delete the DBMSxxxxBIN packaged binary UFDs since they are no longer on the system tape and thus will remain with outdated binaries on your system until manually deleted.

3. Install and share ROAM on your system (see separate ROAM documentation).

4. Restore the UFDs supplied on tape. These may be one or more of the following:

```
DBMSEX     DBMSDEF    DBMSFTN    DBMSF77
DBMSCOB    DBMSCBL    DBMSLGCL
```

5. Run SETUP if you want your configuration options to be non-standard (See following section Introductory Message Control).

6. If you did not run it already as part of step 5 above, issue the following command:

```
CO DBMSEX>DBMS.INSTALL.COMI
```

7. As of Rev. 19.0, the error message files, DAERRS and DBMSE, are copied by the INSTALL procedure from DBMSEX>DBMSLB to the UFD DBMSLB.

8. Share DBMS from the supervisor terminal thus:

```
CO SYSTEM>DBMS.SHARE.COMI
```

   Be sure to put the above share command into PRIMOS.COMI (C_PRMO) in CMDNCO so that DBMS will be shared at any time the system is cold started (See section DMLCP Installation).

9. Finally DELETE the UFD(s) restored from tape.

10. If this is the first time that a ROAM version of DBMS has been installed, complete the steps outlined in the section ROAM Conversion.

Data Administrator Authorization

Every DBMS site must have the following User Profile Groups defined:

.ROAM_ADMIN ROAM Administrators
.DBMS_ADMIN DBMS Data Base Administrators

Persons belonging to the group .DBMS_ADMIN are authorized as valid data base administrators. Without such membership, a user may not use any of the DBACP commands which alter a data base or display sensitive information (such as privacy keys). Persons who also belong to the .ROAM_ADMIN group are classified as privileged administrators. These are data administrators who may bypass the various schema privacy locks when using DBACP. A privileged data base administrator would be responsible for the management and integrity of the DBMS as a whole.

Introductory Message Control

It is possible to inhibit the printing of an introductory message at runtime when DBMS is invoked. To suppress the introductory message, run the CPL procedure SETUP.CPL in DBMSEX>JOBS. This utility asks you if you want the message, makes the appropriate change to VERSIO.FTN, recreates the DMLCP shared segments, and notifies the user to reshare DBMS.

Reloading Products

There are times when a specific subproduct of DBMS needs to be reloaded, that is, the segment directory needs to be created anew. To do this it is possible to use the same job streams that were used in the original building of the components of DBMS. For this to work, you must start with the packaged UFDs delivered on your system tape.

These job streams are found in the JOBS UFDs of the packages to which they belong and they have the names %prod%_LOAD.CPL.

Note that if you have replaced any of the libraries (ULIB, CLIB, ILIB, TEXTED, DMLCP, ASI, or ASG), you should reload all the subproducts that use them in their loading procedures. If you would like to automate the remaining reloads, you can use RESET_UFD.CPL (see section on Files on System Tape).

The load procedures for subproducts produce segment directories in the the UFD DBMSLB within their respective packages. To install a reloaded SEGDIR, you need only copy it up to the top level UFD DBMSLB. The shareable segments produced by LOAD_LIB.CPL should also be promoted to the top level DBMSLB but do not come into use until the next time the DBMS.SHARE.COMI command stream is run (see below).

DMLCP Installation

DMLCP requires the exclusive use of shared segments 2001, 2002, 2003, and 2012 and private segments 4030, 4031, 4032, 4033, and 4034. To install the shared library version of the DML command processor, the following command must be executed from the supervisor terminal:

```
CO SYSTEM>DBMS.SHARE.COMI
```

This command stream installs the DBMS shared library, shares and initializes the DBMS segments, and initializes the Ring 3 semaphores. This command should be incorporated into PRIMOS.COMI (C_PRMO), the command file that is always run after a cold start.

Creation of a DML Application Program

Once a schema and subschema have been written and compiled and the data base files have been allocated with DBACP, you can write application programs for the data base in either COBOL or FORTRAN. The sequence used to transform the source code into executable code is as follows:

1. Preprocess the source code with the host language preprocessor (FDML, CDML, F77DML or CBLDML).

2. Compile the output of the preprocessor (xxxxx.FIN, xxxxx.F77, xxxxx.COBOL, or xxxxx.CBL) with the host language compiler.

3. Link the binary output of the compiler to the DML command processor with the segmented loader SEG.

Command procedures to do these operations with either a COBOL, CBL, FIN or F77 program may be found in UFD DBMSLB called CDML.CPL, CLOAD.CPL, CBLDML.CPL, CBLLOAD.CPL, FDML.CPL, FLOAD.CPL, F77DML.CPL, and F77LOAD.CPL. (The installation utility selects these out of the JOBS UFD of the appropriate package if supplied on the system tape and promotes them to the top level DBMSLB.)

For example, to compile and load a COBOL program called PROG, execute the following command:

```
R DBMSLB>CDML PROG
```

In turn, to compile and load a CBL program called PROG, execute the following command:

```
R DBMSLB>CBLDML PROG
```

Similarly, to compile and load a FORTRAN program called PROG, execute the following command:

```
R DBMSLB>FDML PROG
```

To compile and load a F77 program called PROG, issue the command:

```
R DBMSLB>F77DML PROG
```

The output files created when using CDML.CPL, CBLDML.CPL, FDML.CPL, or F77DML.CPL on the source file PROG are:

```
PROG.LIST - The preprocessor and compiler listings
PROG.BIN  - The binary file output by the compiler
```

The output files from using CLOAD.CPL, CBLLOAD.CPL, or FLOAD.CPL with program PROG are:

```
PROG.MAP - SEG program map
PROG.SEG - The segmented run file
```

The resulting user program is executed with the command:

```
SEG PROG
```

ROAM Conversion

The conversion of existing RAM style schemas to ROAM format is straightforward. No check is made for incomplete transactions before the schema is converted; if there is any doubt the user may perform a DBACP RECOVER SCHEMA operation before doing the conversion.

The conversion of a RAM-style schema to ROAM format involves the following:

1. The new ROAM and DBMS software is installed at the customer site.

2. All data bases should be saved to tape or disk before starting the conversion.

3. The passwords are removed from all the PDBMS UFDs (ROAM does not support passwords).

4. The names in the DALIST are added to the .DBMS_ADMIN and/or .ROAM_ADMIN groups as appropriate. User profiles are updated to reflect new group membership.

5. The new DBUTL command 'ROAM' is used to convert each schema to a ROAM compatible format. All schemas should be converted at once; old style RAM schemas cannot use the new DBACP, SCHED, DMLCP, and so forth. The SDnnnn SEGDIR name in the PDBMS UFD will be changed automatically to <schema-name>.DBMS (the names of all slave SDnnnn's will be changed also). Note that part of the conversion process will be the truncation of the AI and BI files of the schema. Once a schema has been converted to ROAM format, DBACP type AI files cannot be applied against it.

If after-imaging is enabled for the schema, the converted schema has to be saved using one of the ROAM utilities before application programs can be run against it (the ROAM command assigns the newly converted schema a null save date).

One invisible apsect of the conversion process is the resetting of the ROAM shared system-wide transaction number in the ROAM before-image file header. The RAM transaction number in the old per-schema BI file is compared to the new ROAM system-wide transaction number; if it is greater than the ROAM transaction number, it is set to the latter value.

Notes

No DML application programs need to be recompiled to run against the ROAM based schema.

A backward conversion path is available to convert a ROAM file to the old RAM based file. This can be done for schemas that have not been created under a ROAM version of DBMS and for schemas that have not been modified to take advantage of features available only at Rev. 19.3 or later.

The backward conversion process is as follows:

1. Recreate the appropriate PDBMS UFDs. Place the error files, SCHDIR, and so forth, in the master PDBMS UFD. Do not password UFDs yet.

2. An archived ROAM data base must first be restored to active status before it can be converted back to RAM.

3. To start the conversion process, type 'RAM schema_name' in the ROAM based DBUTL.

4. You will be prompted for the volumes to which the master and slave SEGDIRs will reside under RAM. DBMS UFDs (PDBMS) must exist on these volumes. You will have to copy the SEGDIRs to these volumes yourself.

5. As part of the conversion process, the SEGDIR names will be changed to SDNNNN. AI/BI files of minimum size will be allocated. Copy the SEGDIRs to the appropriate DBMS volumes (as entered in point 4). Passwords can now be set for each DBMS UFD.

6. Install the old DBMS software.

7. Increase the size of the BI file (via DBACP).

8. Any application programs that were compiled under ROAM must be recompiled to conform to the old RAM invoke call.

DISCOVER_DBMS

NEW FEATURES AND CHANGES

DISCOVER now supports the F77 and CBL subschema types.

SOFTWARE PROBLEMS FIXED

The RELEASE RECORD command no longer aborts with a NULL_POINTER$ condition being raised (SPAR 3006637).

The OCCURS clause is no longer left off a member of a naming group in a PRISAM record (SPAR 3006716, 3007835).

The LIST RECORDS OF FILE command now appends the .PRISAM suffix to the file name you enter, if it is missing (SPAR 3007834).

The RELEASE RELATION and RELEASE ALL RELATIONS commands now properly clear the current relation name (SPAR 3007837, 3008109).

The SELECT command no longer returns inappropriate data when a null record occurrence (represented by question marks) should be returned (SPAR 3007838).

The SET TERM LENGTH command now affects the length of the update screen forms (SPAR 3007841, 3008110).

When the RECORD COUNT feature of DISCOVER is disabled, the 'Total records selected' message generated by SELECT no longer contains two nonprinting characters that cause the line not to be printed by the spooler when a COMOUTPUT file of the session is printed (SPAR 3007839).

UPDATE commands now default to the current record when there is both a current record and relation (SPAR 3007842).

The USE command no longer displays differing error messages when a record that is already defined is used depending on whether or not the user specified the .PRISAM suffix in the file name (SPAR 3007843).

DISCOVER now aborts any active update transaction when an internal error occurs (SPAR 3007845).

Entering an action code of "A"bort to an update form no longer aborts an active CPL or COMINPUT file (SPAR 3007847).

The error message file is no longer closed inadvertently (SPAR 3007848).

The USE FILE command no longer hangs when the user attempts to use the same record with the same name three times (SPAR 3008306).

Entering the RELEASE ALL RELATIONS command when there are no relations defined no longer causes DISCOVER to abort (SPAR 3009145).

The HELP command within the update screen interface now clears the help text from the screen before returning to the update screen (SPAR 3009148).

The description of the STORE command in the HELP data base has been corrected to show the correct abbreviation.

When the user breaks out of an update command, the verification message now displays the correct command name.

Certain HELP commands no longer cause the HELP subsystem to abort.

Entering a token that starts with an underscore character (_) no longer cause DISCOVER to abort.

JOIN now works when the join key is a complex key containing a group item.

Records that contain no keys may now be used in SELECT commands.

OUTSTANDING PROBLEMS

DISCOVER does not search DBMS records via an available system-owned set when no WHERE clause was specified on the SELECT command (SPAR 2005698).

The SKIP clause sometimes does not work properly (SPAR 3000238).

The SELECT command sometimes returns a different number of data rows from DBMS depending on the data fields retrieved (SPAR 3002177).

DISCOVER generates an internal error when the report width equals the terminal width (SPAR 3002533).

Formats compiled prior to Rev. 19.2.3 are incompatible at the binary level with later releases of DISCOVER (SPAR 3003991).

Schema fields defined as INTEGER*4 are improperly displayed by a CBL subschema that defines them as S9(6) (SPAR 3003993).

PIC 9 fields do not display leading zeros (SPAR 3007122).

PICTURE clauses are not displayed correctly for CBL COMP data items (SPAR 3007248).

Displaying a COBOL record causes an internal DISCOVER table to overflow, while displaying the equivalent FIN record does not (SPAR 3008286).

The LIST RECORDS OF FILE command does not find the desired record when the user-specified pathname already contains the .PRISAM suffix (SPAR 3008307).

Executing the DISPLAY command many times in a session causes DISCOVER to run out of dynamic memory (SPAR 3008561).

ENVIRONMENT

Requires DBMS and ROAM Rev. 19.4, PRISAM Rev. 2.0, and PRIMOS Rev. 19.4.

INSTALLATION AND BUILD PROCEDURES

Standard, except for the following.

Users upgrading from VISTA to DISCOVER must use the DISCOVER_UPGRADE program in the directory DISCOVER_TOOLS to make their VISTA formats, procedures, and abbreviations available to DISCOVER. The instructions in the file DISCOVER_UPGRADE.RUNO in the directory DISCOVER_TOOLS must be followed to insure a successful upgrade from VISTA to DISCOVER. If you are not upgrading from VISTA, you may ignore the preceding paragraph.

DISCOVER uses a configuration file that contains several parameters that define the way DISCOVER interacts with users as well as with the directories that contain the DISCOVER catalogs and the HELP data base. This file is DISCOVER.CONFIG and it is in the directory DISCOVER_DBMS>SYSTEM. It is copied to the directory SYSTEM by the installation procedure. If you have made any changes to this file for prior releases of DISCOVER, please enter them into DISCOVER_DBMS>SYSTEM>DISCOVER.CONFIG before performing the installation or your changes will be lost.

The DISCOVER configuration file, SYSTEM>DISCOVER.CONFIG, consists of 21 lines plus a four line header. Each line must be exactly as described in these instructions or DISCOVER cannot be expected to work properly. The file format consists of the information required by DISCOVER followed by an optional comment on each line. A comment begins with "/*" and ends at the end of the line. The information on each line is as follows:

LINES 1-4: Configuration file header. These lines are ignored by DISCINIT.SAVE; they have been added to help document the file. Do not delete these lines. If they are deleted, DISCINIT still ignores the first four lines of the file and loses necessary information.

LINE 5: The number of characters per line of the TTY display that DISCOVER runs with. This number should be one character less than the actual screen width to avoid unwanted automatic linefeeds (default = 79). Note that this number should be greater than or equal to 71 for optimum performance of DISCOVER.

LINE 6: The number of lines per screen on the TTY that DISCOVER is run with. This number should be one less than the actual screen length to allow for the scrolling prompt (the default = 23).

LINE 7: The number of characters per logical line on the printer that DISCOVER is run with; that is, the number of characters on the line after the printer has inserted its side margins (the default = 132).

LINE 8: The number of lines per logical page on the printer that DISCOVER is run with; that is, the number of lines on the page after the printer has inserted its top and bottom margins (the default = 66).

LINE 9: The maximum number of characters per line to be written to a file when the PRINT command FILE option is used (the default = 132).

LINE 10: The maximum number of lines per page to be written to a file when the PRINT command FILE option is used (the default = 66).

LINE 11: This line is reserved for future expansion and should be left blank.

LINE 12: This line is reserved for future expansion and should be left blank.

LINE 13: The name of the master DISCOVER UFD (the default is DISCOVER*).

LINE 14: The owner password of the master DISCOVER UFD (the default is '').

LINE 15: The owner password of the DISCOVER CATALOG UFD (where the procedures, formats, and abbreviations are stored) (the default is '').

LINE 16: The master UFD of the DISCOVER HELP subsystem files (the default is DISCOVER*).

LINE 17: The owner password of the master HELP UFD (the default is '').

Note

If the default master DISCOVER UFD name (DISCOVER*) is used and the default master DISCOVER HELP Subsystem UFD name (DISCOVER*) is also used, the passwords on this line and line 14 must be the same since the UFDs they apply to are themselves the same.

LINE 18: The DISCOVER HELP UFD (the actual data files of the HELP subsystem reside here) (the default is HELP).

LINE 19: The DISCOVER HELP subsystem UFD owner password (the default is '').

LINE 20: The DISCOVER HELP subsystem topmost level prefix. Since the HELP subsystem prints the actual UFD name of its current location, it deletes the topmost (passworded) UFD names and their passwords from the HELP subsystem header. This prefix replaces the deleted portion (the default is HELP DISCOVER).

LINE 21: The scrolling default. If SCROLL ENABLED is to be the default, set to 1. If SCROLL DISABLED is to be the default, set to 0 (the default is 1).

LINE 22: The number of records retrieved between printing the record count (the default is 1).

Note

If DISCOVER will be used with hard-copy terminals, it is suggested that this be left as the default. To disable the printing of the record count by default, the System Administrator can create a public DISCOVER "STARTUP" procedure containing the command "DISABLE RECORD COUNT". This will avoid the constant overwriting of the record count.

LINE 23: The number of expected tokens to print after a syntax error. When DISCOVER finds a token it didn't expect it prints what it did expect, up to the maximum number of tokens set by this variable. Users with slow or hard-copy terminals may want to set this to lower than the default value to speed up DISCOVER after syntax errors (the default is 48).

LINE 24: The name of the public and/or private DISCOVER procedures automatically executed when entering DISCOVER (the default is STARTUP).

LINE 25: The number associated with the level of update security you wish this installation of DISCOVER to have. Any or all of the update commands can be restricted by changing the value of this parameter (the default is 7, all UPDATE commands enabled). It must be set to the sum of the keys for the functions that you wish to allow from this list:

ERASE = 1, MODIFY = 2, STORE = 4.

PRISAM SELECT Performance Issues

Questions concerning the performance of DISCOVER generally take the form of how a given query should be set up for best performance. Once again there is no one answer. The performance of a query will depend on a number of factors. The best way to address this question is with some general guidelines. Note that some of the guidelines are release dependent and specifics of each guideline may change as development on DISCOVER continues. However, the overall intent of each guideline will not change.

Relation Selection Optimizations: The FROM clause in the SELECT, MODIFY, and ERASE commands specifies the relation that will be processed. Choosing an appropriate relation for a query is important for best performance.

Single record relations are established by the USE command. The command "USE FILE CUSTOMER", assuming that file CUSTOMER is a PRISAM file with a single record of the same name, would establish one single record relation named CUSTOMER. The commands "USE FILE PRODUCT" and "USE FILE ORDER", given similar assumptions, would establish two additional single record relations, PRODUCT and ORDER. Note that single record relations are usually referred to as records in most of the DISCOVER documentation.

Multi-record relations are built by executing one or more JOIN commands. The first JOIN command that is executed on a given relation establishes the base record for the relation, names the relation, and sets up a default processing sequence for the records in the relation. Consider the command:

```
JOIN RECORD CUSTOMER TO RECORD ORDER ~
     ON CUSTOMER-ID = ORDER-CUSTOMER-ID ~
     GIVING RELATION CUSTOMER-ORDER
```

The record named in the "TO RECORD" clause tells DISCOVER that ORDER is the base record for this relation. The relation is named by the "GIVING RELATION" clause. The default processing sequence

is base record ORDER followed by first joined record CUSTOMER. All other records that are part of this relation will be joined directly to the base record or indirectly to the base record through one or more intermediate records.

JOIN commands which use a previously named relation adds records to the named relation. Consider the command:

```
JOIN RECORD PRODUCT TO RELATION CUSTOMER-ORDER ~
     ON PRODUCT-CODE = ORDER-PRODUCT-CODE
```

This command will add another record PRODUCT to an existing relation CUSTOMER-ORDER. The PRODUCT record will be appended to the end of the default processing sequence.

The two JOIN commands above will establish a relation named CUSTOMER-ORDER with a base record of ORDER and a default process sequence of ORDER record, CUSTOMER record, PRODUCT record.

Now consider the commands:

```
JOIN RECORD PRODUCT TO RECORD ORDER ~
     ON PRODUCT-CODE = ORDER-PRODUCT-CODE ~
     GIVING RELATION PRODUCT-ORDER
```

and

```
JOIN RECORD CUSTOMER TO RELATION PRODUCT-ORDER ~
     ON CUSTOMER-ID = ORDER-CUSTOMER-ID
```

These two JOIN commands will establish a relation named PRODUCT-ORDER that is similar to the relation CUSTOMER-ORDER but it will have a default process sequence of ORDER record, PRODUCT record, CUSTOMER record.

If a user followed the command sequence above, there would be three single record relations (CUSTOMER, ORDER, and PRODUCT) and two multi-record relations (CUSTOMER-ORDER and PRODUCT-ORDER) defined to DISCOVER that could be used in the FROM clause. The question is which is the most appropriate for a given query. The following rules should be applied to answer this question:

1. Choose the relation that has the smallest number of records and the smallest PRISAM record descriptions that contain all the data the user is looking for and all of the qualifying conditions that the user is placing on the data.

2. If more than one relation satisfies rule 1, look at the default processing sequence of the relations that satisfy

rule 1 and choose the relation that rejects occurrences of the relation as early as possible along the default processing path if there is a WHERE clause. If there is no WHERE clause, choose the relation that will accept occurrences of the relation as early as possible along the default processing path.

3. Always be aware of implied conditions when using a multi-record relation. "SELECT CUSTOMER FROM CUSTOMER" tells DISCOVER to find each record occurrence in CUSTOMER and copy it into the current table; "SELECT CUSTOMER FROM CUSTOMER-ORDER" tells DISCOVER to find each record occurrence in ORDER, then find the CUSTOMER occurrence that is joined to that ORDER occurrence, and copy that occurrence of CUSTOMER into the current table. The first SELECT will copy one occurrence of each CUSTOMER into the current table. No CUSTOMER occurrence will be skipped and none will be duplicated, assuming none are duplicated in the CUSTOMER file. The second SELECT will copy one occurrence of the joined CUSTOMER into the current table for each occurrence of ORDER in the ORDER file. CUSTOMER will be skipped if there is no joined ORDER. A CUSTOMER that has more than one joined ORDER will be duplicated.

For example, given the relations defined above:

1. Find all of the data about all of the CUSTOMERS.

```
SELECT ALL FROM CUSTOMER
```

The multi-record relations CUSTOMER-ORDER or PRODUCT-ORDER could have also been used, but they do not satisfy rule 1.

2. Find all of the data about PRODUCTS which are out of stock.

```
SELECT ALL FROM PRODUCT WHERE PRODUCT-ONHAND = 0
```

The multi-record relations CUSTOMER-ORDER or PRODUCT-ORDER could have also been used but they do not satisfy rule 1.

3. Find the CUSTOMER-ID and PRODUCT-CODE for all ORDERS.

```
SELECT CUSTOMER-ID, PRODUCT-CODE FROM CUSTOMER-ORDER
```

or

```
SELECT CUSTOMER-ID, PRODUCT-CODE FROM PRODUCT-ORDER
```

Both are equally appropriate because all three records must be processed to satisfy the query and there is no WHERE clause to aid in selecting the best relation.

4. Find the CUSTOMER-ID for all ORDERS.

```
SELECT CUSTOMER-ID FROM CUSTOMER-ORDER
```

CUSTOMER-ORDER is appropriate because of the implied condition. PRODUCT-ORDER could also be used but the PRODUCT would have to be processed when there is no need to (see rule 2). Note that there will be duplicate CUSTOMER-IDs if an occurrence of CUSTOMER has more than one ORDER joined to it. The REMOVE DUPLICATE command can be used to eliminate any duplicates.

5. Find the CUSTOMER-ID for all CUSTOMERS who have ordered PRODUCT AB123.

```
SELECT CUSTOMER-ID FROM PRODUCT-ORDER ~
      WHERE PRODUCT-CODE = 'AB123'
```

PRODUCT-ORDER is appropriate because of the implied condition and DISCOVER can reject occurrences as soon as it has processed ORDER (second on the default process list for PRODUCT-ORDER) without processing the CUSTOMER (third on the list). CUSTOMER-ORDER could also be used but DISCOVER would always process the CUSTOMER (second on the default process list for CUSTOMER-ORDER) before it processed PRODUCT. CUSTOMER occurrences would be processed that had not ordered the product. The same duplicate condition may occur as described in the example above.

Data Ordering Optimizations: DISCOVER will take advantage of the processing sequence of data when it can. Consider the last example above where DISCOVER is asked to find the CUSTOMER-ID of all CUSTOMERS who have ordered PRODUCT AB123. If ORDER was sorted in ORDER-CUSTOMER-ID sequence then in ORDER-PRODUCT-CODE sequence, DISCOVER will be able to take advantage of having already processed CUSTOMER and/or PRODUCT for previous occurrence of ORDER if they have the same ORDER-CUSTOMER-ID and/or ORDER-PRODUCT-CODE as the current occurrence of ORDER. Given the command "SELECT CUSTOMER-ID FROM PRODUCT-ORDER WHERE PRODUCT-CODE = 'AB123'", DISCOVER will process an ORDER occurrence, then the PRODUCT occurrence joined to the ORDER. Assuming that it is indeed product AB123, DISCOVER would then process the CUSTOMER joined to ORDER. Having completed processing of that occurrence of ORDER, DISCOVER will save the PRODUCT and CUSTOMER that it has just processed and proceed to the next occurrence of ORDER. If the next occurrence of ORDER has the

same CUSTOMER and/or PRODUCT joined to it as the previous ORDER had, DISCOVER will perform a substantially shorter processing cycle on CUSTOMER and/or PRODUCT because of the saved data.

PRISAM Key Optimizations: The current release of DISCOVER uses a bottom up access strategy when performing the SELECT, MODIFY, or ERASE command. This strategy entails finding an occurrence of the base record in the relation then finding all additional record occurrence which are joined, directly or indirectly, to that occurrence of the base record. If the command is being performed on a single record relation, there will be no additional records to find. There are two access methods that DISCOVER can use to find occurrences of the base record, a read-first-read-next method or a key-read method. The read-first-read-next method will always work and is used if the key-read method can not be used. It is not the preferred method because it requires finding every record occurrence of the base record via what amounts to a sequential read, from the beginning to the end, of the base file. This is inefficient when there are a large number of base record occurrences and only a few of them satisfy the command. The preferred method is to find base record occurrence via a key read. DISCOVER will use the key-read method when there is a WHERE clause in the command that satisfies the following requirements:

- The WHERE clause must contain at least one simple condition of the form "data-item-name equal literal" where data-item-name is the name of a key in the PRISAM file which contains the base record. The key must have been defined as "DUPLICATES NOT ALLOWED" in the PRISAM Data Definition.

- If there are multiple conditions in the WHERE clause connected by the boolean operator AND (a compound condition), there must not be more than one simple condition containing the data-item-name.

- If there are multiple simple and/or compound conditions in the WHERE clause connected by the boolean connector OR (a complex condition), there must be at least one data-item-name that appears in each of the simple and/or compound conditions that satisfies rules 1 and 2 above.

For example, given a PRISAM file CUSTOMER that contains a key STATE defined as "DUPLICATES NOT ALLOWED" and a data-item AGE that is not a key:

```
SELECT ALL FROM CUSTOMER WHERE AGE = 18
```

Read-first-read-next because rule 1 is not satisfied.

```
SELECT ALL FROM CUSTOMER WHERE STATE = 'MA'
```

Key-read because rule 1 is satisfied and rules 2 and 3 are not broken.

```
SELECT ALL FROM CUSTOMER WHERE STATE = 'NY' AND AGE => 18
```

Key-read because rules 1 and 2 are satisfied and rule 3 is not broken.

```
SELECT ALL FROM CUSTOMER WHERE ~
       (STATE = 'NH' AND AGE => 19) OR ~
       (STATE = 'MA' AND AGE => 20)
```

Key-read because rules 1 and 2 are satisfied and rule 3 is not broken.

```
SELECT ALL FROM CUSTOMER WHERE ~
       (STATE = 'MA' AND AGE => 20) OR (AGE > 21)
```

Read-first-read-next because rule 3 is not satisfied.

Command Optimizations: A sequence of simple DISCOVER commands will normally provide better overall system performance than one compound command, especially in a multi-user environment involving large PRISAM files. The command sequence of "SELECT ALL FROM CUSTOMER" followed by a DISPLAY command will normally take less elapsed time and make better use of system resources than the command "SELECT AND DISPLAY ALL FROM CUSTOMER". This will be especially apparent if there are several other users on the system competing with DISCOVER for system resources and the DISCOVER user take time to peruse each screen of data before scrolling to the next screen.

DBMS SELECT Performance Issues

The purpose of this section is to describe the behavior of the ASG (Access Strategy Generator) thus forming a model of its behavior. This discussion applies to all versions of DBMS prior to and including Rev. 19.4.

Because the ASG is a significant software undertaking, this document contains a considerable amount of densely-packed information about its behavior. A good understanding of the behavior of the ASG can be achieved through careful study of this document. The audience for this document is assumed to be fairly familiar with the facilities provided by DISCOVER/DBMS, especially the SELECT command. In particular, readers are assumed to be familiar with SELECT syntax and purpose, and with DBMS features used in schema and subschema design, including the concept of the virtual record. Many terms, however, are used in this document and not defined elsewhere in DBMS or DISCOVER documentation. Please refer to the glossary at the end of this discussion for the definitions of unfamiliar terms.

All examples in this document utilize a single schema and subschema contained in the appendices of this document.

The behavior of the ASG can be described by outlining the processing and stating the rules which the ASG follows in constructing a strategy. A rule is composed of two parts: conditions and results. The general format of a rule is:

> IF <a set of conditions about the query and/or the data base is true>,
>
> THEN <a set of results occurs>.

Conditions: The conditions in each ASG rule are composed of simple conditions about either the query (SELECT command) being processed or about the data base (subschema or schema) being accessed. These simple conditions all have one thing in common: each simple condition potentially affects the access strategy chosen by the ASG.

The set of conditions which form the IF part of an ASG rule is made up of simple conditions about either the SELECT command or about the data base.

SELECT Command Conditions: These conditions fall into two categories: characteristics of target lists and characteristics of WHERE clause predicates.

Target List Characteristics: The characteristics of the target list which are important to the ASG refer to how many records and which records of the virtual record structure are chosen to be part of the target list. Each subschema record that contains an item that is referenced in the target list must be retrieved by the access strategy when materializing the portion of the virtual record being selected. In general, the fewer different subschema records referenced in the target list and the closer these records are to the base record, the more efficient the access strategy and the performance of the query.

WHERE Clause Predicate Characteristics: The WHERE clause of a SELECT command specifies the restrictions placed on the virtual records to be retrieved by the command. Thus, the contents of the WHERE clause affects the access strategy chosen. Among the important factors are:

- Whether the items referenced are access keys of the data base
- What kind of keys they are
- The logical operators connecting the WHERE clause predicates and their precedence
- The number of different subschema records containing the items referenced in the WHERE clause

Data Base Conditions: These conditions relate to the structure of the schema and subschema which define the data base being accessed. Many factors are involved, among which are:

- The number of different record types that comprise the virtual record
- The position of each record type in the subschema structure
- The presence and number of access keys useful as entry points to the structure (These access keys to entry point record types are called record keys.)
- The locations of these keys within the subschema structure
- The presence, number, and types of sets relating record types within the subschema structure

Sizes of Areas and Sets: Actual statistics of areas and sets, such as the number of buckets in an area, the number of record occurrences in an area, or the number of members per owner a set contains, are unknown to the ASG and have no current effect on access strategy generation. However, the ASG does attempt to estimate the cost of executing a given potential top-down access strategy, using these statistics, in order to guide it in choosing the most efficient strategy. Hypothetical values for these statistics are used in access strategy generation.

Results: The output of an execution of the ASG on a SELECT command is a generated access strategy. This strategy is then executed against the data base by the Access Strategy Interpreter (ASI). The result of an ASG rule specifies characteristics useful in generating an access strategy.

Types of Strategies: There are two basic types of access strategies:

1. Bottom-Up: The bottom-up access strategy scans each bucket of each area containing occurrences of the base record type of the virtual record. If a record occurrence satisfies the predicates specified on it in the WHERE clause, then all owners that are required to satisfy additional predicates or materialize target list items are found. These find-owner operations are done for all necessary record types, utilizing the sets and structures specified in the subschema.

2. Top-Down: The top-down access strategy utilizes the subschema structure and the WHERE clause predicate to limit the number of record occurrences accessed in each record type of the virtual record. Records accessed must include those containing items used in the WHERE clause predicates or in the target list. In a top-down strategy, the data base structure is navigated from a record type at the top of the schema structure, called an entry point, downward via the appropriate sets until the base record is reached, with record occurrences qualified or materialized along the way. Record types that are not along the access path from the entry point to the base record are qualified and materialized as needed by finding the owner via sets which relate these records to records that are along the main access path.

The ASG frequently uses several entry points to retrieve the desired virtual records, utilizing different top-down substrategies.

Expected Access Strategy Performance: In general, top-down access strategies perform better than bottom-up access strategies. The main component of this performance is the actual number of disk accesses which are made in reading area buckets, sets, and calc files. For the bottom-up strategies, reading every bucket of an area will take a fairly long time, especially with large areas. However, areas that are small may make the bottom-up strategy more efficient in some cases. Top-down strategies have a much better chance of performing better since efforts are made to restrict the number of records actually accessed by using sets and calc files.

Note that since actual values for the sizes of the areas, sets, and so on, are not known to the ASG, some access strategies which are deemed most efficient by the ASG may in fact not be the most efficient available. This is an important area for future improvement.

ASG Behavior: The purpose of the ASG is to determine and generate the most efficient access strategy available to satisfy the SELECT command. The input to the ASG are descriptions of the virtual record, the target list, and the WHERE clause, all of which are specified or implied in the SELECT command. The output of the ASG is the set of instructions used by DBMS to execute the access strategy and produce individual virtual records which match the WHERE clause predicate. These

instructions are interpreted and executed by a portion of DBMS called the Access Strategy Interpreter (ASI).

The processing that the ASG performs in generating a strategy is organized into two major sections. The first section analyzes the input provided and the structure of the virtual record and data base to determine the best access strategy available. The second section generates the instructions which implement the access strategy selected.

As stated above, conditions and results are combined to form the rules that the ASG follows in producing access strategies. The actual rules used by the ASG are contained in the following subsections.

Determining the Access Strategy: Determining the access strategy to be executed to satisfy a query is the heart of the ASG. It is where the decisions are made regarding the most efficient strategy to be chosen. This processing is decomposed into several steps:

1. Gather schema information: Scan the data base schema to gather relevant information about the items specified in the WHERE clause and target list items.

   Rule:

   IF no WHERE clause is specified on the SELECT command,

   THEN a bottom-up strategy is generated.

   Rule:

   IF no simple predicate in the WHERE clause on the SELECT command references the initial item of any key,

   THEN a bottom-up strategy is generated.

2. Normalize the WHERE clause predicate: The WHERE clause of a SELECT command can specify the same predicate in many different ways. In order to determine whether a SELECT command can be processed using a top-down strategy, the ASG converts the WHERE clause into a standard or normal form.

   The smallest component of a normalized predicate is known as a simple predicate. It has the form 'operand operator operand' (such as, CUSTOMER-ID = 3968). The next level of aggregation within a normalized predicate is called the record predicate. Simple predicates referring to items within the same record type can be collected together into a unit, the record predicate.

For example:

```
CUSTOMER-ID = 3968 OR CUSTOMER-NAME = 'SMITH'
```

Record predicates are next grouped together with the AND operator. These groupings of record predicates are called conjunctive groups (CGs). For example:

```
(CUSTOMER-ID = 3968 OR CUSTOMER-NAME = 'SMITH') AND
 ORDER-DATE > 830401 AND PRODUCT-ONHAND > 100
```

Finally, the entire WHERE clause is organized into disjunctive normal form (DNF). DNF has the general form:

```
(record-pred AND ... AND record-pred) OR ... OR
(record-pred AND ... AND record-pred).
```

In the above, all CGs are ORed together to form the entire WHERE clause predicate. For example:

```
((CUSTOMER-ID = 3968 OR CUSTOMER-NAME = 'SMITH') AND
  ORDER-DATE > 830401 AND PRODUCT-ONHAND > 100) OR
 (ITEM-QUANTITY > 1000 AND ORDER-SALESPERSON = 'SCHMIDT')
```

If utilizing a top-down strategy is possible, each conjunctive group in the normalized WHERE clause predicate generates its own top-down substrategy.

The first step of normalization is to use DeMorgan's Laws to distribute all negation operators inward to the simple predicates. For example, if ^ (CUSTOMER-ID < 3968) is specified in the WHERE clause, the normalized version is CUSTOMER-ID >= 3968.

The second step in the normalization process is to convert the whole WHERE clause predicate to DNF. Note that collections of simple predicates that form record predicates are not split up during the conversion to DNF.

For example, the normal form of:

```
CUSTOMER-ID = 3968 AND
(ORDER-DATE > 830401 OR ITEM-QUANTITY > 1000)
```

is:

```
(CUSTOMER-ID = 3968 AND ORDER-DATE > 830401) OR
(CUSTOMER-ID = 3968 AND ITEM-QUANTITY > 1000)
```

whereas:

```
(CUSTOMER-ID = 3968 OR CUSTOMER-NAME = 'SMITH')
 AND ORDER-DATE > 830401
```

is already in normal form.

The third step is to convert each record predicate to DNF as well.

3. Examine the normalized record predicates: Each normalized record predicate must exhibit certain characteristics for a top-down access strategy to be considered. The rules in this section describe those characteristics and apply to each record predicate.

Rule:

> IF at least one simple predicate in a conjunctive group of a record predicate specifies an access key,
>
> AND EITHER the access key is a full or partial sort/search key,
>
> OR the access key is a full calc key and the predicate operator is equality,
>
> THEN that conjunctive group is called amenable.

For Example:

```
SELECT ORDER-RECORD FROM ORDER-RECORD WHERE ~
ORDER-DATE = 830401 AND ORDER-SHIP-BY > 830908 OR ~
ORDER-NUMBER > 12345
```

In this example, the entire WHERE clause is a single record predicate that contains two conjunctive groups. In the first group, both items of ORDER-RECORD participate in a search list

of the SYSTEM-SRTD-ORDER-2 set. They thus constitute an access key that is a full search key. In the second group, ORDER-NUMBER is the calc key for ORDER-RECORD. Both conjunctive groups are therefore amenable.

Rule:

IF more than one simple predicate in a conjunctive group of a record predicate qualifies that conjunctive group as amenable,

THEN only one record key is chosen to make that conjunctive group amenable.

The record key selected is the one predicted to be the most selective over all the record occurrences of that record type. In general, equality predicates are more selective than predicates containing inequality operators, such as GE or NE, and two inequality predicates on the same item specifying an upper and lower bound on the desired values are more selective than just one.

For Example:

```
SELECT ORDER-RECORD FROM ORDER-RECORD WHERE ~
ORDER-DATE = 830401 AND ORDER-SHIP-BY = 830908 AND ~
ORDER-NUMBER > 12345
```

This example contains just one record predicate with one conjunctive group. There are three possible record keys which make this conjunctive group amenable:

- Through the ascending key of SYSTEM-SRTD-ORDER-1
- Through the ascending key of SYSTEM-SRTD-ORDER-2
- Through the second search key of SYSTEM-SRTD-ORDER-2

In the first two cases, the items involve inequality operators. In the third case, the two items specify equality operators. Thus, the second search key is chosen as the most selective record key to make this conjunctive group amenable.

Rule:

IF all conjunctive groups of a normalized record predicate are amenable,

THEN that normalized record predicate is also amenable.

4. Examine the normalized WHERE clause predicate: The normalized WHERE clause predicate must also exhibit certain characteristics for a top-down access strategy to be considered. The rules in this section describe those characteristics.

Rule:

IF any conjunctive group contains no amenable record predicates,

THEN a bottom-up access strategy will be generated for this SELECT command.

For Example:

```
SELECT ORDER-RECORD FROM ORDER-RECORD WHERE ~
CUSTOMER-NAME = 'DVORAK VIOLA CO.' AND ~
ORDER-SHIP-BY > 830908 OR ORDER-NUMBER > 12345
```

In this example, there are two conjunctive groups. In the second one, after the OR, ORDER-NUMBER is the record key through set SYSTEM-SRTD-ORDER-1. In the first conjunctive group, however, neither item is an access key or a calc key, and therefore this conjunctive group contains no amenable record predicates. A bottom-up access strategy is generated.

Rule:

IF a record predicate of a conjunctive group is amenable, AND the access key that made this conjunctive group amenable is a record key,

THEN that record type is a candidate entry point for a top-down substrategy to be generated.

Rule:

IF at least one record predicate of a conjunctive group of the normalized WHERE clause predicate refers to a record which is a candidate entry point for a top-down substrategy,

THEN that conjunctive group is a candidate for a top-down substrategy.

For Example:

```
SELECT ORDER-RECORD FROM ORDER-RECORD WHERE ~
CUSTOMER-NAME = 'DVORAK VIOLA CO.' AND ~
ORDER-NUMBER > 12345
```

The WHERE clause in this example contains two record predicates. Only the second of which is amenable. Since ORDER-NUMBER is the record key through set SYSTEM-SRTD-ORDER-1, ORDER-RECORD is a candidate entry point record for a top-down access strategy. Therefore, this conjunctive group (the only one in the query) is a candidate for a top-down substrategy.

Rule:

IF all conjunctive groups of the normalized WHERE clause predicate are candidates for a top-down substrategy,

AND at least one entry point of a top-down substrategy is not the base record accessed through a full sequential scan of a system-owned set,

Then a top-down access strategy will be generated for this SELECT command.

For Example:

```
SELECT ORDER-RECORD FROM ORDER-RECORD WHERE ~
ORDER-DATE > 830908 OR CUSTOMER-NAME = ~
'DVORAK VIOLA CO.' AND ORDER-NUMBER > 12345
```

In this example, the second conjunctive group is already known to be a candidate for a top-down substrategy. The first conjunctive group is also since ORDER-DATE is a partial search key of the SYSTEM-SRTD-ORDER-2 set, thus making it amenable and a record key. Thus, two top-down access substrategies will be generated for this query.

Rule:

IF at least one conjunctive group of the normalized WHERE clause predicate is NOT a candidate for a top-down substrategy,

THEN a bottom-up access strategy will be generated for this SELECT command.

For Example:

```
SELECT ORDER-RECORD FROM ITEM-RECORD WHERE ~
ITEM-QUANTITY > 100 OR CUSTOMER-NAME = ~
'DVORAK VIOLA CO.' AND ORDER-NUMBER > 12345
```

The second conjunctive group is a candidate for a top-down access substrategy. The first group is amenable since ITEM-QUANTITY is the descending key of the PRODUCT-ITEM set. However, it is not a record key. Thus, the conjunctive group of which it is the only component is not a candidate for a top-down access substrategy and, therefore, a bottom-up strategy will be generated.

5. Select the best substrategies: Under the assumption that a top-down access strategy will be employed, the ASG must now choose the best one. One top-down access substrategy is necessary for each conjunctive group and these substrategies are independent of each other, although they might select duplicate virtual record occurrences. In fact, the ASG may need to choose between several possible substrategies for each conjunctive group, considered separately.

Rule:

IF a record predicate of a member record type specifies a (partial) sort/search key of a set that lies along the access path from the entry point record to the base record, (called a set key)

THEN that (partial) sort/search key is utilized to access occurrences of that member record type during a top-down access substrategy in order to reduce the number of the record occurrences accessed from the owner record type.

Rule:

IF there is only one candidate entry point for a top-down substrategy,

THEN that entry point will be chosen as the entry point for that top-down substrategy.

For Example:

```
SELECT PRODUCT-ONHAND, ITEM-QUANTITY FROM ITEM-RECORD ~
WHERE PRODUCT-CODE = 'AJ104580' AND ITEM-QUANTITY > 100
```

The WHERE clause contains only one conjunctive group since there are no OR operators. There are two record predicates since PRODUCT-CODE is in PRODUCT-RECORD and ITEM-QUANTITY is in ITEM-RECORD. In fact, these are the only two record types utilized in this query since the target list items are contained in these two records also. The PRODUCT-RECORD type utilizes PRODUCT-CODE both as a calc key and as an ascending key through the SYSTEM-SRID-PRODUCT-1 set. Since equality is the predicate operator in the WHERE clause, the calc key will be used to access this entry record type. The ITEM-RECORD type has no key items which can be used to access each ITEM-RECORD record occurrence. Thus, PRODUCT-RECORD is the only candidate entry point for a top-down substrategy. Note that since there is only one conjunctive group, there is only one top-down substrategy to be generated. ITEM-QUANTITY will be used to restrict access to record occurrences in ITEM-RECORD that are owned by the occurrences of PRODUCT-RECORD which satisfy the record predicate. The PRODUCT-ITEM set is used for this purpose.

Rule:

IF there is more than one candidate entry point for a top-down substrategy, over a given conjunctive group,

THEN only one entry point is chosen to begin the substrategy for this conjunctive group. The entry point selected is the one predicted to have the least execution cost of all entry point candidates. The cost estimations for each entry point are done by complex calculations, based on the predicted, and hypothetical, statistics relating to the data base and the normalized WHERE clause predicate. In general, the choice exhibits these characteristics:

- The entry point with the most selective record predicate is chosen
- The entry point with the most selective predicates on the sets encountered along the access path to the base record is chosen
- The entry point farthest from the base record is chosen

There is no clear priority given to these guidelines. For example:

```
SELECT PRODUCT-DESCRIPTION FROM ITEM-RECORD WHERE ~
ORDER-NUMBER > 10000 AND PRODUCT-CODE = 'AJ104523'
```

There are three record types involved in this query: ORDER-RECORD, ITEM-RECORD, and PRODUCT-RECORD. ORDER-NUMBER is the sort list for the set SYSTEM-SRTD-ORDER-1, thereby making the ORDER-RECORD a candidate entry point. PRODUCT-CODE is both the calc key for PRODUCT-RECORD and the sort list for the SYSTEM-SRTD-PRODUCT-1 set, thus making PRODUCT-RECORD also a candidate entry point. The calc key is used since equality is the predicate operator. The ASG chooses the PRODUCT-RECORD as the entry point for this query since the equality operator is more selective than the greater-than operator. For example:

```
SELECT PRODUCT-DESCRIPTION, PRODUCT-ONHAND FROM ~
ITEM-RECORD WHERE ORDER-NUMBER = 10000 AND ~
PRODUCT-CODE = 'AJ104523' AND ITEM-QUANTITY > 100
```

Again there are three record types involved in this query: ORDER-RECORD, ITEM-RECORD, and PRODUCT-RECORD. ORDER-NUMBER is the sort list for the set SYSTEM-SRTD-ORDER-1 and the calc key for ORDER-RECORD, thereby making the ORDER-RECORD a candidate entry point using the calc key. PRODUCT-CODE is both the calc key for PRODUCT-RECORD and the sort list for the SYSTEM-SRTD-PRODUCT-1 set, thus making PRODUCT-RECORD also a candidate entry point using the calc key. In addition, ITEM-QUANTITY is the sort key of the PRODUCT-ITEM set. Therefore, since the predicates on the two entry record types are equally selective, the ASG chooses the PRODUCT-RECORD as the entry point for this query since the ITEM-QUANTITY predicate is a more selective access path than the ORDER-ITEM set, which has no sort keys at all. For example:

```
SELECT CUSTOMER-NAME FROM ITEM-RECORD WHERE ~
ORDER-NUMBER > 10000 AND PRODUCT-CODE > 'AJ104523' AND ~
CUSTOMER-ID > 1000
```

Again there are three record types involved in this query: CUSTOMER-RECORD, ORDER-RECORD, and PRODUCT-RECORD. ITEM-RECORD is also involved since it is the base record for the virtual record. CUSTOMER-NUMBER is the sort list for the set SYSTEM-SRTD-CUSTOMER-1, thereby making CUSTOMER-RECORD a candidate entry point using that set. ORDER-NUMBER is the sort list for the set SYSTEM-SRTD-ORDER-1, thereby making ORDER-RECORD a candidate entry point using that set. And

PRODUCT-CODE is the sort list for the SYSTEM-SRID-PRODUCT-1 set, thus making PRODUCT-RECORD also a candidate entry point using that set. The ASG chooses the CUSTOMER-RECORD as the entry point for this query since each record predicate is equally selective, no additional selectivity is present along any of the connecting sets, and CUSTOMER-RECORD is farthest from the base record.

6. Generating the Access Strategy: The navigation that the two basic access strategies utilize has been described above. The bottom-up strategy makes one pass over the data base qualifying and materializing virtual records. The top-down strategy is actually several serial strategies, each with its own pass over the data base. Duplicate qualified virtual record occurrences are prevented.

Glossary

The reader is assumed to be familiar with the concepts and terminology of CODASYL data bases, virtual record structures, and SELECT commands. This section defines some additional terms that are used in this document.

Access Key: A record key or a set key.

Access Path: A portion of a schema structure (sets, records, and calc files) which provides the means to navigate from an entry point to the base record of a virtual record and to all record types required for materialization or qualification.

Amenable: A (portion of a) predicate in a SELECT command is amenable if it specifies access keys that permit a top-down strategy to utilize those keys to limit the number of record occurrences accessed for qualification.

ASG - Access Strategy Generator: The portion of DBMS intended for DISCOVER/DBMS support that determines the most effective means of navigating the subschema in order to satisfy a query against a virtual record.

ASI - Access Strategy Interpreter: The portion of DBMS intended for DISCOVER/DBMS support that executes the access strategy generated by the the ASG.

Conjunction: A predicate consisting of the logical AND of one or more subpredicates; for example, A AND B AND C.

Conjunctive Group: A conjunction of record predicates forming the main components of the disjunctive normal form of a predicate.

Disjunction: A predicate consisting of the logical OR of one or more subpredicates; for example, A OR B OR C.

Disjunctive Normal Form (DNF): A canonical form for predicates. A predicate is in DNF if it is the disjunction of one or more conjunctions. Every predicate consisting of ANDs, ORs, and NOTs of simple predicates has an equivalent DNF predicate.

Entry Point: Given a predicate against a virtual record structure, an entry point for that predicate is a record type in the virtual record structure that is used as the starting point for navigation of the structure to identify those virtual records satisfying the predicate.

Materialization: The goal of an access strategy that retrieves a set of record occurrences forming a virtual record in order to satisfy the target list of a SELECT command.

Partial Key (Value): For access keys that are composed of several data items in a record, a partial key is one where not every data item is specified in the query. At least the first item is specified and possibly others in increasing sequence, as long as there is at least one data item which is not specified.

Predicate: A statement of conditions that restricts the virtual records selected for retrieval in a SELECT command. The statement is expressed using logical operators.

Qualification: The goal of an access strategy that retrieves record occurrences and checks simple predicate conditions on items contained in those record occurrences for the purpose of deciding whether a candidate virtual record satisfies the WHERE clause conditions of a SELECT command.

Qualified Record Type: A record type against which there is a record predicate.

Record Key: A sort key or a search key of a system-owned set whose related record type has AUTOMATIC and MANDATORY membership in that system-owned set, or a calc key of a record.

Record Predicate: A predicate referencing items of a single record type.

Rule: In this document, a precise description of some aspect of the behavior of the ASG.

Set Key: A sort key or a search key of a set relating two record types that lie along the access path from the entry point record and the base record.

Simple Condition: A statement in this document about characteristics of the SELECT command or the data base that contains neither conjunctions nor disjunctions of other conditions.

Simple Predicate: A predicate that does not involve the logical operators AND, OR, and NOT; for example, LASTNAME='SMITH'.

Example Schema

```
SCHEMA NAME IS AW2-SCHEMA.
AREA NAME IS ORDER-AREA.
RECORD NAME IS CONTROL-RECORD;
    LOCATION MODE IS CALC USING CONTROL-KEY
    DUPLICATES ARE NOT ALLOWED.
/*      01  CONTROL-RECORD.*/
        01  CONTROL-KEY; INTEGER*2.
        01  CONTROL-LAST-CUSTOMER-ID; PICTURE '9(4)'.
        01  CONTROL-LAST-ORDER-NUMBER; PICTURE '9(6)'.
RECORD NAME IS CUSTOMER-RECORD;
    LOCATION MODE IS CALC USING CUSTOMER-ID
    DUPLICATES ARE NOT ALLOWED.
/*      01  CUSTOMER-RECORD.*/
        01  CUSTOMER-ID; INTEGER*2.
        01  CUSTOMER-NAME; PICTURE 'X(20)'.
        01  CUSTOMER-ADDRESS; PICTURE 'X(20)'.
        01  CUSTOMER-CITY; PICTURE 'X(12)'.
        01  CUSTOMER-STATE-COUNTY; PICTURE 'X(7)'.
        01  CUSTOMER-ZIP-POST-CODE; PICTURE 'X(7)'.
        01  CUSTOMER-COUNTRY; PICTURE 'X(12)'.
RECORD NAME IS ORDER-RECORD;
    LOCATION MODE IS CALC USING ORDER-NUMBER
    DUPLICATES ARE NOT ALLOWED.
/*      01  ORDER-RECORD.*/
        01  ORDER-NUMBER; INTEGER*4.
        01  ORDER-DATE; PICTURE '9(6)'.
        01  ORDER-SALESPERSON; PICTURE 'X(20)'.
        01  ORDER-SHIP-BY; PICTURE 'X(20)'.
RECORD NAME IS ITEM-RECORD;
    LOCATION MODE IS VIA ORDER-ITEM SET.
/*      01  ITEM-RECORD.*/
        01  ITEM-QUANTITY; PICTURE '9(3)'.
        01  ITEM-BACKORDERED; PICTURE '9(3)'.
RECORD NAME IS PRODUCT-RECORD;
    LOCATION MODE IS CALC USING PRODUCT-CODE
    DUPLICATES ARE NOT ALLOWED.
/*      01  PRODUCT-RECORD.*/
        01  PRODUCT-CODE; PICTURE 'X(8)'.
        01  PRODUCT-DESCRIPTION; PICTURE 'X(20)'.
        01  PRODUCT-PRICE; PICTURE '9(2)V99'.
        01  PRODUCT-ONHAND; PICTURE '9(4)'.
        01  PRODUCT-LOCATION.
            02  PRODUCT-ROW; PICTURE 'X(2)'.
            02  PRODUCT-BIN; PICTURE '9(2)'.
```

```
SET NAME IS CUSTOMER-ORDER;
    ORDER IS SORTED;
    OWNER IS CUSTOMER-RECORD.
    MEMBER IS ORDER-RECORD MANDATORY AUTOMATIC;
    ASCENDING KEY IS ORDER-DATE DUPLICATES ALLOWED.
    SET OCCURRENCE SELECTION IS THRU CURRENT OF SET.
SET NAME IS ORDER-ITEM;
    ORDER IS LAST;
    OWNER IS ORDER-RECORD.
    MEMBER IS ITEM-RECORD MANDATORY AUTOMATIC;
    SET OCCURRENCE SELECTION IS THRU CURRENT OF SET.
SET NAME IS PRODUCT-ITEM;
    ORDER IS SORTED;
    OWNER IS PRODUCT-RECORD.
    MEMBER IS ITEM-RECORD OPTIONAL AUTOMATIC;
    DESCENDING KEY IS ITEM-QUANTITY DUPLICATES ALLOWED;
    SET OCCURRENCE SELECTION IS THRU CURRENT OF SET.
SET NAME IS SYSTEM-SRTD-CUSTOMER-1;
    ORDER IS SORTED;
    OWNER IS SYSTEM.
    MEMBER IS CUSTOMER-RECORD MANDATORY AUTOMATIC;
    ASCENDING KEY IS CUSTOMER-ID DUPLICATES NOT ALLOWED.
SET NAME IS SYSTEM-SRTD-ORDER-1;
    ORDER IS SORTED;
    OWNER IS SYSTEM.
    MEMBER IS ORDER-RECORD MANDATORY AUTOMATIC;
    ASCENDING KEY IS ORDER-NUMBER DUPLICATES NOT ALLOWED.
SET NAME IS SYSTEM-SRTD-ORDER-2;
    ORDER IS SORTED;
    OWNER IS SYSTEM.
    MEMBER IS ORDER-RECORD MANDATORY AUTOMATIC;
    ASCENDING KEY IS ORDER-NUMBER,
        ORDER-DATE DUPLICATES NOT ALLOWED;
    SEARCH KEY IS ORDER-SALESPERSON DUPLICATES ALLOWED;
    SEARCH KEY IS ORDER-DATE,
        ORDER-SHIP-BY DUPLICATES ALLOWED.
SET NAME IS SYSTEM-ORDER-LAST;
    ORDER IS LAST;
    OWNER IS SYSTEM.
    MEMBER IS ORDER-RECORD MANDATORY AUTOMATIC.
SET NAME IS SYSTEM-SRTD-PRODUCT-1;
    ORDER IS SORTED;
    OWNER IS SYSTEM.
    MEMBER IS PRODUCT-RECORD MANDATORY AUTOMATIC;
    ASCENDING KEY IS PRODUCT-CODE DUPLICATES NOT ALLOWED;
    SEARCH KEY IS PRODUCT-ROW,
        PRODUCT-BIN DUPLICATES ALLOWED.
END SCHEMA.
```

Example COBOL Subschema

```
SUBSCHEMA NAME IS SUB-1 OF SCHEMA AW2-SCHEMA.
AREA SECTION
COPY ALL AREAS.
RECORD SECTION
01  CUSTOMER-RECORD.
    02  CUSTOMER-ID; PICTURE 9(4).
    02  CUSTOMER-NAME; PICTURE X(20).
    02  CUSTOMER-ADDRESS; PICTURE X(20).
    02  CUSTOMER-CITY; PICTURE X(12).
    02  CUSTOMER-STATE-COUNTY; PICTURE X(7).
    02  CUSTOMER-ZIP-POST-CODE; PICTURE X(7).
    02  CUSTOMER-COUNTRY; PICTURE X(12).
01  ORDER-RECORD.
    02  ORDER-NUMBER; PICTURE 9(6).
    02  ORDER-DATE; PICTURE 9(6).
    02  ORDER-SALESPERSON; PICTURE X(20).
    02  ORDER-SHIP-BY; PICTURE X(20).
01  ITEM-RECORD.
    02  ITEM-QUANTITY; PICTURE 9(3).
    02  ITEM-BACKORDERED; PICTURE 9(3).
01  PRODUCT-RECORD.
    02  PRODUCT-CODE; PICTURE X(8).
    02  PRODUCT-DESCRIPTION; PICTURE X(20).
    02  PRODUCT-PRICE; PICTURE 9(2)V99.
    02  PRODUCT-ONHAND; PICTURE 9(4).
    02  PRODUCT-LOCATION.
        03  PRODUCT-ROW; PICTURE X(2).
        03  PRODUCT-BIN; PICTURE 9(2).
SET SECTION
    COPY ALL SETS.
```

DISCOVER_PRISAM

SOFTWARE PROBLEMS FIXED

The RELEASE RECORD command no longer aborts with a NULL_POINTER$ condition being raised (SPAR 3006637).

The OCCURS clause is no longer left off a member of a naming group in a PRISAM record (SPAR 3006716, 3007835).

The LIST RECORDS OF FILE command now appends the .PRISAM suffix to the file name you enter, if it is missing (SPAR 3007834).

The RELEASE RELATION and RELEASE ALL RELATIONS commands now properly clear the current relation name (SPAR 3007837, 3008109).

The SELECT command no longer returns garbage data when a null record occurrence (represented by question marks) should be returned (SPAR 3007838.

The SET TERM LENGTH command now affects the length of the update screen forms (SPAR 3007841, 3008110).

When the RECORD COUNT feature of DISCOVER is disabled, the "Total records selected" message generated by SELECT no longer contains two non-printing characters that cause the line not to be printed by the spooler when a COMOUTPUT file of the session is printed (SPAR 3007839).

UPDATE commands now default to the current record when there is both a current record and relation (SPAR 3007842).

The USE command no longer displays differing error messages, when a record that is already defined is used, depending on whether or not the user specified the .PRISAM suffix in the file name (SPAR 3007843).

DISCOVER now aborts any active update transaction when an internal error occurs (SPAR 3007845).

Entering an action code of "A"bort to an update form no longer aborts an active CPL or COMINPUT file (SPAR 3007847).

The error message file is no longer closed inadvertantly (SPAR 3007848).

The USE FILE command no longer hangs when the user attempts to use the same record with the same name three times (SPAR 3008306).

Entering the RELEASE ALL RELATIONS command when there are no relations defined no longer causes DISCOVER to abort (SPAR 3009145).

The HELP command within the update screen interface now clears the help text from the screen before returning to the update screen (SPAR 3009148).

The description of the STORE command in the help data base has been corrected to show the correct abbreviation.

When the user breaks out of an update command, the verification message now displays the correct command name.

Certain HELP commands no longer cause the HELP subsystem to abort.

Entering a token that starts with an underscore character (_) no longer cause DISCOVER to abort.

JOIN now works when the join key is a complex key containing a group item.

Records that contain no keys may now be used in SELECT commands.

OUTSTANDING PROBLEMS

The SKIP clause sometimes does not work properly (SPAR 3000238).

DISCOVER generates an internal error when the report width equals the terminal width (SPAR 3002533).

Formats compiled prior to Rev. 19.2.3 are incompatible at the binary level with later releases of DISCOVER (SPAR 3003991).

PIC 9 fields do not display leading zeros (SPAR 3007122).

The LIST RECORDS OF FILE command does not find the desired record when the user-specified pathname already contains the '.PRISAM' suffix (SPAR 3008307).

Executing the DISPLAY command many times in a session causes DISCOVER to run out of dynamic memory (SPAR 3008561).

ENVIRONMENT

Requires ROAM Rev. 19.4, PRISAM Rev. 2.0, and PRIMOS Rev. 19.4.

INSTALLATION AND BUILD PROCEDURES

Standard except for the following:

Users upgrading from VISTA to DISCOVER must use the DISCOVER_UPGRADE program in the directory DISCOVER_TOOLS to make their VISTA formats, procedures, and abbreviations available to DISCOVER. The instructions in the file DISCOVER_UPGRADE.RUNO in the directory DISCOVER_TOOLS must be followed to insure a successful upgrade from VISTA to DISCOVER. If you are not upgrading from VISTA, you may ignore the preceding paragraph.

DISCOVER uess a configuration file that contains several parameters that define the way DISCOVER interacts with users as well as with the directories that contain the DISCOVER catalogs and HELP data base. This file is DISCOVER.CONFIG and it is in the directory DISCOVER_PRISAM>SYSTEM. It is copied to the directory SYSTEM by the installation procedure. If you have made any changes to this file for prior releases of DISCOVER, please enter them into DISCOVER_PRISAM>SYSTEM>DISCOVER.CONFIG before performing the installation or your changes will be lost.

The DISCOVER configuration file, SYSTEM>DISCOVER.CONFIG, consists of 21 lines plus a 4 line header. Each line must be exactly as described in these instructions or DISCOVER cannot be expected to work properly. The file format consists of the information required by DISCOVER followed by an optional comment on each line. A comment begins with "/*" and ends at the end of the line. The information on each line is as follows:

LINES 1-4: Configuration file header. These lines are ignored by DISCINIT.SAVE. They have been added to help document the file. Do not delete these lines. If they are deleted, DISCINIT will still ignore the first 4 lines of the file and will lose necessary information.

LINE 5: The number of characters per line in the TTY display that DISCOVER is run with. This number should be one character less than the actual screen width to avoid unwanted automatic linefeeds (the default = 79). Note that this number should be greater than or equal to 71 for optimum performance of DISCOVER.

LINE 6: The number of lines per screen in the TTY display that DISCOVER is run with. This number should be one less than the actual screen length to allow for the scrolling prompt (the default = 23).

LINE 7: The number of characters per logical line on the printer that DISCOVER is run with; that is, the number of characters on the line after the printer has inserted its side margins (the default = 132).

LINE 8: The number of lines per logical page on the printer that DISCOVER is run with; that is, the number of lines on the page after the printer has inserted its top and bottom margins (the default = 66).

LINE 9: The maximum number of characters per line to be written to a file when the PRINT command FILE option is used (the default = 132).

LINE 10: The maximum number of lines per page to be written to a file when the PRINT command FILE option is used (the default = 66).

LINE 11: This line is reserved for future expansion and should be left blank.

LINE 12: This line is reserved for future expansion and should be left blank.

LINE 13: The name of the master DISCOVER UFD (the default = 'DISCOVER*').

LINE 14: The owner password of the master DISCOVER UFD (the default = '').

LINE 15: The owner password of the DISCOVER CATALOG UFD (where the procedures, formats and abbreviations are stored) (the default = '').

LINE 16: The master UFD of the DISCOVER HELP subsystem files (the default = 'DISCOVER*').

LINE 17: The owner password of the master HELP UFD (the default = '').

Note

If the default master DISCOVER UFD name (DISCOVER*) is used and the default master DISCOVER HELP Subsystem UFD name (DISCOVER*) is also used, then the passwords on this line and line 14 must be the same since the UFDs they apply to are themselves the same.

LINE 18: The DISCOVER HELP UFD (the actual data files of the HELP subsystem reside here) (the default = 'HELP').

LINE 19: The DISCOVER HELP subsystem UFD owner password (the default = '').

LINE 20: The DISCOVER HELP subsystem topmost level prefix. Since the HELP subsystem prints the actual name of

the UFD where it is currently located, it deletes the topmost (passworded) UFD names and their passwords from the HELP subsystem header. This prefix replaces the deleted portion (the default = 'HELP DISCOVER').

LINE 21: The scrolling default. If 'SCROLL ENABLED' is to be the default, set to 1. If 'SCROLL DISABLED' is to be the default, set to 0 (the default = 1).

LINE 22: The number of records retrieved between printing the record count (the default = 1).

Note

If DISCOVER will be used with hard-copy terminals, it is suggested that this number be left at the default. To disable the printing of the record count by default, the System Administrator can create a public DISCOVER "STARTUP" procedure containing the command "DISABLE RECORD COUNT". This will avoid the constant overwriting of the record count.

LINE 23: The number of expected tokens to print after a syntax error. When DISCOVER finds a token it did not expect, it prints what it did expect, up to the maximum number of tokens set by this variable. Users with slow or hard-copy terminals may want to set this to lower than the default value to speed up DISCOVER after syntax errors (the default = 48).

LINE 24: The name of the public and/or private DISCOVER procedures automatically executed when entering DISCOVER (the default = STARTUP).

LINE 25: The number associated with the level of update security you wish this installation of DISCOVER to have. Any or all of the update commands can be restricted by changing the value of this parameter (the default is 7, all UPDATE commands enabled). It must be set to the sum of the keys for the functions that you wish to allow from this list:

ERASE = 1, MODIFY = 2, STORE = 4.

PRISAM SELECT Performance Issues

Questions concerning the performance of DISCOVER generally take the form of how a given query should be set up for best performance. Once again there is no one answer. The performance of a query will depend on a number of factors. The best way to address this question is with some general guidelines. Note that some of the guidelines are release dependent and specifics of each guideline may change as development on DISCOVER continues. However, the overall intent of each guideline will not change.

Relation Selection Optimizations: The FROM clause in the SELECT, MODIFY, and ERASE commands specifies the relation which will be processed. Choosing an appropriate relation for a query is important for best performance.

Single record relations are established by the USE command. The command "USE FILE CUSTOMER", assuming that file CUSTOMER is a PRISAM file with a single record of the same name, would establish one single record relation named CUSTOMER. The commands "USE FILE PRODUCT" and "USE FILE ORDER", given similar assumptions, would establish two additional single record relations, PRODUCT and ORDER. Note that single record relations are usually referred to as records in most of the DISCOVER documentation.

Multi-record relations are built by executing one or more JOIN commands. The first JOIN command that is executed on a given relation establishes the base record for the relation, names the relation, and sets up a default processing sequence for the records in the relation. Consider the command:

```
JOIN RECORD CUSTOMER TO RECORD ORDER ~
    ON CUSTOMER-ID = ORDER-CUSTOMER-ID ~
     GIVING RELATION CUSTOMER-ORDER
```

The record named in the "TO RECORD" clause tells DISCOVER that ORDER is the base record for this relation. The relation is named by the "GIVING RELATION" clause. The default processing sequence is base record ORDER followed by first joined record CUSTOMER. All other records that are part of this relation will be joined directly to the base record or indirectly to the base record through one or more intermediate records.

JOIN commands which use a previously named relation will add records to the named relation. Consider the command:

```
JOIN RECORD PRODUCT TO RELATION CUSTOMER-ORDER ~
     ON PRODUCT-CODE = ORDER-PRODUCT-CODE
```

This command adds another record PRODUCT to an existing relation CUSTOMER-ORDER. The PRODUCT record is appended to the end of the default processing sequence.

The two JOIN commands above establish a relation named CUSTOMER-ORDER with a base record of ORDER and a default process sequence of ORDER record, CUSTOMER record, PRODUCT record.

Now consider the commands:

```
JOIN RECORD PRODUCT TO RECORD ORDER ~
     ON PRODUCT-CODE = ORDER-PRODUCT-CODE ~
     GIVING RELATION PRODUCT-ORDER
```

and

```
JOIN RECORD CUSTOMER TO RELATION PRODUCT-ORDER ~
     ON CUSTOMER-ID = ORDER-CUSTOMER-ID
```

These two JOIN commands establish a relation named PRODUCT-ORDER that is similar to the relation CUSTOMER-ORDER but it has a default process sequence of ORDER record, PRODUCT record, CUSTOMER record.

If a user followed the command sequence above, there would be three single-record relations (CUSTOMER, ORDER, and PRODUCT) and two multi-record relations (CUSTOMER-ORDER and PRODUCT-ORDER) defined to DISCOVER which could be used in the FROM clause. The question is which is the most appropriate for a given query. The following rules should be applied to answer this question.

1. Choose the relation that has the smallest number of records and the smallest PRISAM record descriptions that contain all the data the user is looking for and all of the qualifying conditions that the user is placing on the data.

2. If more than one relation satisfies rule 1, look at the default processing sequence of the relations that satisfy rule 1 and choose the relation that will reject occurrences of the relation as early as possible along the default processing path if there is a WHERE clause. If there is no WHERE clause, choose the relation that will accept occurrences of the relation as early as possible along the default processing path.

3. Always be aware of implied conditions when using a multi-record relation. "SELECT CUSTOMER FROM CUSTOMER" tells DISCOVER to find each record occurrence in CUSTOMER and copy it into the current table. "SELECT CUSTOMER FROM CUSTOMER-ORDER" tells DISCOVER to find each record occurrence in ORDER, then find the CUSTOMER occurrence that is joined to that ORDER occurrence,

and copy that occurrence of CUSTOMER into the current table. The first SELECT will copy one occurrence of each CUSTOMER into the current table. No CUSTOMER occurrence will be skipped and none will be duplicated, assuming none are duplicated in the CUSTOMER file. The second SELECT will copy one occurrence of the joined CUSTOMER into the current table for each occurrence of ORDER in the ORDER file. CUSTOMERS will be skipped if there is no joined ORDER and CUSTOMER that have more than one joined ORDER will be duplicated.

For example, given the relations defined above:

1. Find all of the data about all of the CUSTOMERS.

```
SELECT ALL FROM CUSTOMER
```

The multi-record relations CUSTOMER-ORDER or PRODUCT-ORDER could have also been used, but they do not satisfy rule 1.

2. Find all of the data about PRODUCTS which are out of stock.

```
SELECT ALL FROM PRODUCT WHERE PRODUCT-ONHAND = 0
```

The multi-record relations CUSTOMER-ORDER or PRODUCT-ORDER could have also been used but they do not satisfy rule 1.

3. Find the CUSTOMER-ID and PRODUCT-CODE for all ORDERS.

```
SELECT CUSTOMER-ID, PRODUCT-CODE FROM CUSTOMER-ORDER
```

or

```
SELECT CUSTOMER-ID, PRODUCT-CODE FROM PRODUCT-ORDER
```

Both are equally appropriate because all three records must be processed to satisfy the query and there is no WHERE clause to aid in selecting the best relation.

4. Find the CUSTOMER-ID for all ORDERS.

```
SELECT CUSTOMER-ID FROM CUSTOMER-ORDER
```

CUSTOMER-ORDER is appropriate because of the implied condition. PRODUCT-ORDER could also be used but the PRODUCT would have to

be processed when there is no need to (see rule 2). Note that there will be duplicate CUSTOMER-IDs if an occurrence of CUSTOMER has more than one ORDER joined to it. The REMOVE DUPLICATE command can be used to eliminate any duplicates.

5. Find the CUSTOMER-ID for all CUSTOMERS who have ordered PRODUCT AB123.

```
SELECT CUSTOMER-ID FROM PRODUCT-ORDER ~
    WHERE PRODUCT-CODE = 'AB123'
```

PRODUCT-ORDER is appropriate because of the implied condition and DISCOVER can reject occurrences as soon as it has processed ORDER (second on the default process list for PRODUCT-ORDER) without processing the CUSTOMER (third on the list). CUSTOMER-ORDER could also be used but DISCOVER would always process the CUSTOMER (second on the default process list for CUSTOMER-ORDER) before it processed PRODUCT. CUSTOMER occurrences would be processed that had not ordered the product. The same duplicate condition may occur as described in the example above.

Data Ordering Optimizations: DISCOVER will take advantage of the processing sequence of data when it can. Consider the last example above where DISCOVER is asked to find the CUSTOMER-ID of all CUSTOMERS who have ordered PRODUCT AB123. If ORDER was sorted in ORDER-CUSTOMER-ID sequence then in ORDER-PRODUCT-CODE sequence, DISCOVER will be able to take advantage of having already processed CUSTOMER and/or PRODUCT for previous occurrences of ORDER if they have the same ORDER-CUSTOMER-ID and/or ORDER-PRODUCT-CODE as the current occurrence of ORDER. Given the command "SELECT CUSTOMER-ID FROM PRODUCT-ORDER WHERE PRODUCT-CODE = 'AB123'", DISCOVER will process an ORDER occurrence, then the PRODUCT occurrence joined to the ORDER and, assuming that it is indeed product AB123, DISCOVER would process the CUSTOMER joined to ORDER. Having completed processing of that occurrence of ORDER, DISCOVER will save the PRODUCT and CUSTOMER that it has just processed and proceed to the next occurrence of ORDER. If the next occurrence of ORDER has the same CUSTOMER and/or PRODUCT joined to it as the previous ORDER had, DISCOVER will perform a substantially shorter processing cycle on CUSTOMER and/or PRODUCT because of the saved data.

PRISAM Key Optimizations: The current release of DISCOVER uses a bottom up access strategy when performing the SELECT, MODIFY, or ERASE command. This strategy entails finding an occurrence of the base record in the relation then finding all additional record occurrences which are joined, directly or indirectly, to that occurrence of the base record. If the command is being performed on a single record relation, there will be no additional records to find. There are two access methods that DISCOVER can use to find occurrences of the base

record, a read-first-read-next method or a key-read method. The read-first-read-next method will always work and is used if the key-read method can not be used. It is not the preferred method because it requires finding every record occurrence of the base record via what amounts to a sequential read, from the beginning to the end, of the base file. This is inefficient when there are a large number of base record occurrences and only a few of them satisfy the command. The preferred method is to find base record occurrence via a key read. DISCOVER will use the key-read method when there is a WHERE clause in the command that satisfies the following requirements:

- The WHERE clause must contain at least one simple condition of the form "data-item-name equal literal" where data-item-name is the name of a key in the PRISAM file which contains the base record. The key must have been defined as "DUPLICATES NOT ALLOWED" in the PRISAM Data Definition.

- If there are multiple conditions in the WHERE clause connected by the boolean operator AND (a compound condition), there must not be more than one simple condition containing the data-item-name.

- If there are multiple simple and/or compound conditions in the WHERE clause connected by the boolean connector OR (a complex condition), there must be at least one data-item-name that appears in each of the simple and/or compound conditions that satisfies rules 1 and 2 above.

  For example, given a PRISAM file CUSTOMER that contains a key STATE that has been defined as "DUPLICATES NOT ALLOWED" and a data-item AGE that is not a key:

```
SELECT ALL FROM CUSTOMER WHERE AGE = 18
```

  Read-first-read-next because rule 1 is not satisfied.

```
SELECT ALL FROM CUSTOMER WHERE STATE = 'MA'
```

  Key-read because rule 1 is satisfied and rules 2 and 3 are not broken.

```
SELECT ALL FROM CUSTOMER WHERE STATE = 'NY' AND AGE => 18
```

Key-read because rules 1 and 2 are satisfied and rule 3 is not broken.

```
SELECT ALL FROM CUSTOMER WHERE ~
       (STATE = 'NH' AND AGE => 19) OR ~
       (STATE = 'MA' AND AGE => 20)
```

Key-read because rules 1 and 2 are satisfied and rule 3 is not broken.

```
SELECT ALL FROM CUSTOMER WHERE ~
       (STATE = 'MA' AND AGE => 20) OR (AGE > 21)
```

Read-first-read-next because rule 3 is not satisfied.

Command Optimizations: A sequence of simple DISCOVER commands will normally provide better overall system performance than one compound command, especially in a multi-user environment involving large PRISAM files. The command sequence of "SELECT ALL FROM CUSTOMER" followed by a DISPLAY command will normally take less elapsed time and make better use of system resources than the command "SELECT AND DISPLAY ALL FROM CUSTOMER". This will be especially apparent if there are several other users on the system competing with DISCOVER for system resources and the DISCOVER user take time to peruse each screen of data before scrolling to the next screen.

DISCOVER_UPGRADE

This document describes DISCOVER_UPGRADE, a conversion tool that is to be used to upgrade a site from VISTA to DISCOVER.

USAGE

```
DISCOVER_UPGRADE [ -PARTITION, -PART, -PAR, -P partition-name
                   -NO_DELETE, -NODEL, -ND, -N
                   -COMOUT, -COMO, -C [ comout-pathname ] ]
```

This command creates the DISCOVER* UFD on the same partition where the VISTA* UFD resides unless a partition name is specified with the -PARTITION option, in which case the partition given will be used. Do not put angle brackets around the partition.

DISCOVER_UPGRADE also copies the public and user catalogs from the VISTA* UFD to the DISCOVER* UFD. Unless the -NO_DELETE option is given, this program deletes the VISTA* UFD as well as all other portions of VISTA that reside in the directories CMDNCO and SYSTEM (VISTA.SAVE in CMDNCO, the VI20 share files in SYSTEM, and so on).

If the -COMOUT option is given, DISCOVER_UPGRADE keeps a comoutput file of the session in DISCOVER_UPGRADE.COMO in your current UFD or in a treename that you specify.

OTHER CONSIDERATIONS

Run this program before installing DISCOVER on your system. You must use DISCOVER_DBMS.INSTALL.COMI or DISCOVER_PRISAM.INSTALL. COMI to install DISCOVER instead of DISCOVER_DBMS.INITINSTALL.COMI or DISCOVER_PRISAM.INITINSTALL.COMI.

You must have at least ALURW access to the partition on which you want to install DISCOVER as well as at least DALURW access to the UFDs SYSTEM, CMDNCO, and the partition on which VISTA is presently installed.

MIDASPLUS

NEW FEATURES AND CHANGES

Support of More File Units

With PRIMOS Dynamic File Units at Rev. 19.4, many more file units per user are available. MIDASPLUS takes advantage of this by supporting up to 512 file units per user. The FUNITS configuration directive can still be used to limit the number of file units used for SUBFILES by a single user. The default for FUNITS is 256, the maximum is 512, and the minimum is 2. Please note that if FUNITS is set to 40, for example, then a single user may still be using 512 file units. However, no more than 40 of those units are for SUBFILES. The remaining units used are for main SEGMENT DIRECTORIES. This means that a user can have more than 40 MIDASPLUS files open even though the FUNITS directive is set to 40.

SOFTWARE PROBLEMS FIXED

Per user common information is now re-initialized at each program re-entry (SPAR 3007917).

Note

For those customers who may have taken advantage of the earlier functionality allowing files to be left open across programs, a switch has been added to the MPLUS.CONFIG file. However, the use of this switch is not recommended because it allows MIDASPLUS files to be closed without updating the MIDASPLUS per user and system shared information. The new configuration directive is:

```
INIT_USER_COMMON  [ ON  ] /* default, initialize
                  [ OFF ] /* don't re-initialize on
                              re-entry.
```

User common will be re-initialized at each new program (command level) change unless explicitly switched off.

Attempting to add duplicate secondary indices to a file where they are not allowed now gives the correct error message (SPAR 3006360).

MPACK performance has been restored to its previous level (SPAR 3007453).

When NTYFM$ is called for a file to which the user only has LUR rights, MIDASPLUS will allow access to that file for read only use. This change allows read only access of MIDAS files by COBOL users. Note that if a file is open for read only but the user has write access on the directory the file is in, MIDASPLUS will allow the user to write to the file. To avoid this, use ACLs to restrict file access or let MIDASPLUS open the file with OPENM$ and a K$READ key. When a user aborts using UPDAT$ because a file is read-only, UPDAT$ now unlocks the record before exiting (SPAR 3005723).

MDUMP now works on remote files (SPAR 3002951).

MDUMP no longer dumps previously deleted secondary keys (SPAR 3004147).

MPACK no longer loops infinitely on an empty secondary index (SPAR 3001371).

DELET$ will now release the record lock if the record is deleted (SPAR 3002360).

Midasplus no longer uses only one slave per remote file for all users on the system. Each user now has a unique slave.

OUTSTANDING PROBLEMS

Automatic cleanup hinders the ability to debug a language error because it cleans up the stack, preventing the use of DMSTK (SPAR 3001525).

BILD$R seems to handle negative bit strings incorrectly (SPAR 3003147).

Deferred quits interrupt MPLUSCLUP (SPAR 3007075).

Counters are not updated when the last user has read-only rights (SPAR 3007516).

MDUMP does not seem to properly dump all of the records in a large MIDASPLUS file that has been corrupted (SPAR 3009782).

MPLUSCLUP appears to unconditionally set TERM -BREAK on (SPAR 3010388).

A damaged file shows negative numbers with the USAGE option of CREATK (SPAR 3010527).

ENVIRONMENT

At Rev. 19.4, MIDASPLUS has the following dependencies:

BASICV	19.4
COBOL	19.4
POWERPLUS	19.4
PRIMOS	19.4

INSTALLATION AND BUILD PROCEDURES

The MIDASPLUS installation procedures are standard using the two command files described below.

MIDASPLUS.INSTALL.COMI: COMINPUT file that places all the MIDASPLUS files in the appropriate system directories. This file is similar to the standard master disk install files.

MIDASPLUS.SHARE.COMI: COMINPUT file that properly shares and initializes MIDASPLUS on the system. This file is similar to the standard master disk share files.

The initial directory comes with everything built and ready for installation. This version of MIDASPLUS includes all the online and the offline routines, and the MIDAS utilities: CREATK, KBUILD, KIDDEL, MDUMP, MPACK, and SPY. In addition, both an R-mode library, KIDALB, and a synonym library, VKDALB, are built for compatibility with previous MIDAS/MIDASPLUS releases.

If you desire to use any of the MIDASPLUS configuration directives (see below) you should create a file called MPLUS.CONFIG in SYSTEM containing the required configuration directives. Note that you do not need to use config directives; they are optional.

It is recommended that MIDASPLUS.SHARE.COMI be placed in the PRIMOS.COMI (C_PRMO) system initialization file so that MIDASPLUS is shared and initialized at each system cold start.

Password Directories

Beginning at Rev. 19.3, a change in the scheme for preventing concurrent file access by a MIDASPLUS utility user and a MIDASPLUS runtime library user requires that any MIDASPLUS file on which a utility will be used be in a directory protected by ACLs and that its parent directory be protected by ACLs also. The utilities will only work in a passworded directory if there is no owner password on the current directory or on the parent directory.

This new scheme reads and then changes the RWLOCK on the MIDASPLUS file to EXCLUSIVE before a utility is allowed access to the file. Any user who wishes to use the MIDASPLUS utilites on an existing file must have the following ACL rights:

- L, U, P on the current directory
- L, U on the parent directory

Configuration Parameters

MIDASPLUS has available configuration parameters that may be set at system initialization. MIDASPLUS is initialized by the IMIDASPLUS command contained in the MIDASPLUS.SHARE.COMI COMINPUT file. The IMIDASPLUS command will set some configuration parameters as directed in a configuration file, if specified, and will make the appropriate initialization call to the MIDASPLUS library. Below is a description of the configuration file and how to utilize it.

IMIDASPLUS allows a configuration file treename to be given on the command line or will optionally display a list of the available configuration parameters if -HELP is supplied on the command line. If a treename is not specified, IMIDASPLUS will assume that the configuration file to be utilized is file MPLUS.CONFIG in the current directory. If the MPLUS.CONFIG file does not exist, then IMIDASPLUS will assume that all system configuration defaults are to be assumed and will continue the initialization.

The configuration file may contain a number of directives that specify MIDASPLUS configuration parameters. These parameters are contained in standard text, one per line, within the file. If an error is encountered with a particular directive, IMIDASPLUS will print an error message, assume the default, and continue the initialization.

The available configuration directives are described in the paragraphs that follow. Any parameter not specified by a directive will assume its default value.

SEMAPHORE <semaphore-number>

Specifies the PRIMOS semaphore, given by <semaphore-number>, utilized for user synchronization. The default for <semaphore-number> is -14.

DEBUG [ON / OFF]

Controls MIDASPLUS debug execution and print options for developer debug. The default for this parameter is OFF. This parameter sets debug control on a system wide basis. It may be set on a per user basis by calls to MSGCTL.

FUNITS <max number of file units>

Specifies the maximum number of file units per user that MIDASPLUS will use for MIDASPLUS subfiles. (This number does not include file units that are used for main MIDASPLUS segment directories.) The default is 256 and the maximum is 512. The value specified should not be less than the maximun number of MIDASPLUS segment directories a user is likely to have open at a time. In most cases, the value should be at least four times that number. This will allow for four subfiles per MIDASPLUS file to be open.

TIMEOUT <seconds>

Specifies the number of seconds that any user will wait for some internal resource (such as locks, buffers) before MIDASPLUS assumes that the system is hung and aborts the current operation. The argument <seconds> should be a non-negative integer expressing the maximum number of seconds to wait for any resource. If <seconds> is zero, no timeout will occur and the user will wait indefinitely. The default value is 300 seconds or 5 minutes.

PRINT_ERROR [ON / OFF]

Specifies whether MIDASPLUS should print error messages when fatal errors occur. If ON is given, MIDASPLUS will print the MIDAS error code for any fatal type of error encountered. If a PRIMOS system call error was encountered, it will also print the system error message. Specifying OFF will cause MIDASPLUS to not print any error messages. The default for this parameter is ON. This parameter sets the error print control on a system wide basis. It may additionally be controlled on a per user basis by calls to MSGCTL.

BUFFERS <buffer-count>

Specifies the number of internal file buffers that MIDASPLUS is to utilize. The value of <buffer-count> must be from 2 to 64, inclusive. Giving a low value will utilize less working-set but increase the number of system I/Os and user wait time for a buffer. Giving a large value will decrease the user wait times, but utilize more working-set. The default value for <buffer-count> is 64. It is recommended that this parameter not be changed except on systems with few MIDASPLUS users and limited memory.

REMOTE_TRANSMIT [ON / OFF]

When set ON, allows outgoing MIDASPLUS access to remote files. When set OFF disallows outgoing remote calls. The default for this parameter is ON.

REMOTE_RECEIVE [ON / OFF]

When set ON, allows incoming requests from other network nodes for access to MIDASPLUS files on this system to be processed. When set OFF, incoming requests are denied. The default for this parameter is ON.

REPORT_DUPS [ON / OFF]

When set ON, the reporting of the existence of duplicate entries, by returning a value of one in the returned status code (word one of the ARRAY), is enabled. When set OFF, a status code of one is never returned. This facility is useful since MIDAS was inconsistent in returning the duplicate indicator and many applications which consider a non-zero returned status to be an error might no longer operate in the same fashion with MIDASPLUS. The default value for this parameter is ON. The parameter sets a system wide control on the returning of duplicate indicators. It may be controlled on a per user basis by calls to MSGCTL.

REPORT_LOCKED [ON / OFF]

When set ON, the reporting of locked records when performing read operations is enabled (that is FIND$, NEXT$ operations). The fact that the record, which has been read, is locked is indicated by the value of Bit 5 in array word 13 being set to 1 (as in MIDAS). This should not be confused with Bit 1 of array word 10, which is set to 1 to indicate successful operation of a locked record call. When set to OFF (the default) Bit 5 of array word 13 is always set to zero, regardless of the record's locked status. This directive sets the system wide state for locked record reporting. It may be set on a per-user basis by MSGCTL.

SPY_FNAMES [OFF / ON]

When set ON, MIDASPLUS will save the filenames of open MIDAS files for SPY to use when displaying which files have records locks. When set OFF, will not save the filenames. The default OFF.

PIELIB

At Rev. 19.4, PIELIB is used in creating the DBMS preprocessors CDML, CBLDML, FDML, and F77DML together with versions of preprocessors required by PRIMEWAY. Complete information may be found in the update package DBMS Data Manipulation Language Reference Guide (UPD5308-1LA).

NEW FEATURES AND CHANGES

Two additional statement formats are provided for FIND and FETCH. These are as follows:

Format 7

```
{ FIND  }
{ FETCH } rec-name VIA [ [CURRENT OF] SET set-name
                       [ SUBKEYED dbid-1 [, dbid-2] ... ]
```

Format 8

```
{ FIND  } { NEXT  }
{ FETCH } { PRIOR } RECORD [rec-name] OF CURRENT LIST OF SET set-name
```

In addition, F77 support is provided for DBMS in the form of F77DML.

Kernel support of CBL has been enhanced to handle unlimited levels of qualification and unlimited subscripts. Checking of the correct syntax for CBL (or COBOL) is now left to the appropriate compiler.

Kernel support for PL/1 (including PLP/SPL) now supports qualified data names and unlimited subscripts.

SOFTWARE PROBLEMS FIXED

Underscore is now a legal character in CBL and COBOL identifiers (TPAR 2076).

The OF/IN qualifiers of COBOL and CBL must be acceptable in any case (TPAR 2159, 2353).

A qualified and/or subscripted identifier is now handled correctly when split across multiple lines (TPAR 2232, 2233).

The character sequences 'QUAL NAME' and 'EOT' that appear in error messages have been expanded to more meaningful messages (TPAR 2648).

The censoring of statements by the PRIMEWAY preprocessor has been amended to ensure that it works on punctuated statements (TPAR 2664).

INSTALLATION AND BUILD PROCEDURES

Standard.

Note

The UFD structure has been slightly reorganized to place some additional files in INTCOM*.

PRIME POWER

NEW FEATURES AND CHANGES

POWER Passwords

The POWER system passwords have been removed from the PASSWD file in POWER*. This will not affect the PASSWD SYSTEM command. A utility to convert to the new password format is included in POWERPLUS>TOOLS. This utility will be run by the POWERPLUS install file.

Private Procedure Files

POWER now supports PUBLIC and PRIVATE PROCEDURE files. PRIVATE PROCEDURES are stored in the UFD specified during the PROCEDURE CREATE command and are only accessible by the user that created the procedure.

PROC CREATE: The format of the PROC CREATE command has been changed to allow the user to specify a public or private procedure as follows:

```
PROC CREATE PUBLIC  or  PROC CREATE PRIVATE
```

PRIVATE is the default CREATE mode. CREATE PRIVATE asks for the following information:

```
ENTER DIRECTORY (UFD) NAME :
ENTER DIRECTORY PASSWORD   :
ENTER PROCEDURE NAME       :
ENTER PROCEDURE DESCRIPTION:
```

'*' is accepted as a valid directory name.

CREATE PUBLIC creates procedure files in the POWRCM UFD.

PROC LIST: The format of this command is as follows:

```
PROC LIST  or  PROC LIST ALL
```

PROC LIST lists all procedure files created by the current user. PROC LIST ALL lists all public procedure files.

PROC DELETE: PROC DELETE allows the user to delete any private procedure file created under his or her LOGIN ID or any public procedure file. Existing procedure files do not have an associated LOGIN ID and can be deleted by any user.

PROCS Created By Other Commands: The CREATE, FORM, REPORT, and SCREEN commands allow the user to save responses in a procedure file. When the user requests that responses be saved, the following prompts are issued:

```
DO YOU WANT TO CREATE A PRIVATE PROC? :
ENTER DIRECTORY(UFD) NAME             :
ENTER DIRECTORY PASSWORD              :
ENTER PROCEDURE NAME                  :
ENTER PROCEDURE DESCRIPTION           :
```

If the user chooses to create a public procedure, the command omits the prompts for directory name and directory password.

Converting To The New PROC Format: This enhancement requires a change in the structure of the procedure dictionary file. When Rev. 19.4 POWER is installed, it will be necessary to run a utility to convert POWR##. Existing procedure files do not have associated LOGIN IDs and are all marked as public files. The utility to convert to the new PROC format is in POWERPLUS>TOOLS and is run by the POWERPLUS install command file.

DISPLAY USING SCREEN

The command format is as follows:

```
DISPLAY [<setN>] USING SCREENXX
```

setN is an optional set name or set number and SCREENXX is a valid screen name.

DISPLAY USING SCREEN displays the first record in the set on the specified screen. The bottom line of the screen displays:

```
ENTER COMMAND ("H" FOR HELP):                    RECORD: 1
```

The current record number is always displayed on the bottom of the screen. The cursor is always positioned after the "ENTER COMMAND ... " prompt.

The following are the commands to be entered:

Command	Meaning
H	The HELP command displays the available commands: END, REDRAW, NEXT, PREVIOUS, TOP, BOTTOM, <N> (Absolute, +/- (Relative)
END	Clear the screen and return to POWER command level
REDRAW	Redisplay the screen and the current record
NEXT	Display the next record in the set
PREVIOUS	Display the previous record in the set
TOP	Display the first record in the set
BOTTOM	Display the last record in the set
<N>	Display the Nth record in the set
+N or -N	Display the Nth relative record in the set
	All commands can be abbreviated to one letter.

DESTROY -DELETE Option

A -DELETE option has been added to the DESTROY command. The following command removes the file from the POWER dictionary and also deletes the disk file:

```
DESTROY <filename> <password> -DELETE
```

If the file was created with '*' as the directory name, the user must be attached to the directory containing the file in order to delete it.

Table Skip

When adding records to a file with an interactive ADD command, all or part of a table field may be blank-filled by entering '$N' as the next value in the table. '$N' causes the add to skip to the next field in the record.

LIST SYSTEM Command

The LIST SYS or LIST SYSTEM command allows the System Administrator to list all POWER files, both public and private.

Null Commands

POWER no longer displays 'INVALID COMMAND' when the user enters a blank command.

Display FILLER Fields

The DESCRIPTORS command now displays FILLER fields.

Numeric Field Names

The CREATE and CREATE CHANGE commands now accept the following field type names:

REAL*8	or	NUM
REAL	or	NUM2
INT	or	NUM3
INT*4	or	NUM4
DEC	or	NUM5
COMP-3	or	NUM6

LIST VALIDATION

The LIST VALIDATION command lists all of the validated descriptors and their associated validation files for the currently selected file. The command syntax is as follows:

LIST VAL or LIST VALIDATION

This command will display the following:

DESCRIPTOR	FILE NAME
**********	*********

DESCRIPTOR is the name of the descriptor being validated and FILE NAME is the name of the file that contains its validation table. If FILE NAME is blank, the descriptor is validated through a range check.

RANGE Function

The RANGE function displays the following values for a specified descriptor in the current or specified set:

LOWEST VALUE

LOWER QUARTILE

MEDIAN

MEAN

UPPER QUARTILE

HIGHEST VALUE

NUMBER OF ENTRIES

These have the following definitions:

MEDIAN	value at position (N+1)/2
LOWER QUARTILE	value at position (N+2)/4
UPPER QUARTILE	value at position (3N+2)/4.
MEAN	the average of the N set elements

The number of entries is N, the number of elements in the set. RANGE values can be obtained in two ways: by invoking the RANGE function for a particular descriptor and through the REPORTS: function.

Interactive RANGE Function

A file must be selected and a set created from that file. The command:

RANGE [set name] <descriptor>

displays the following for the specified descriptor:

```
RANGE OF <descriptor> IS
**********************

MIN      LQ      MEDIAN      MEAN      UQ      MAX      NO
---      --      ------      ----      --      ---      --
```

All values except number of entries, are printed to four decimal places. RANGE can only be used on numeric descriptors.

Invoking RANGE From a REPORT: The REPORT CREATE dialogue asks "DO YOU WANT AVERAGE OR RANGE" for any numeric descriptor. The user may respond with any one of the following:

YES	(AVERAGE ONLY)
BOTH	(AVERAGE AND RANGE)
RANGE	(RANGE ONLY)
NO	(NO AVERAGE OR RANGE)

If RANGE was selected, the RANGE values are displayed at the end of the REPORT.

PST100 & PT200

POWER now supports the PST100 and the PT200 terminals. The WREN must be index 9 in the POWER terminal table and the PT200 must be index 10. Therefore, TERM** and TERM## should be copied from POWERPLUS>TOOLS.

Note

When using the PT200, it is recommended that the user does not change screen size within a POWERPLUS session.

Terminals Supported

POWER now supports terminals which do cursor addressing in YX order as well as those which do addressing in XY order. Support has also been added for terminals which do not utilize cursor address codes. Because of this new functionality, changes have been made to the command files that configure the terminal drivers. Therefore the user should note the changes made to the following command files located in POWERPLUS>TOOLS.

PT45.COMI	OWL.COMI	WREN.COMI	
FOX.COMI	HARD.COMI	INFOTON4.COMI	
PT65.COMI	REG100.COMI	REGENT.COMI	PT200.COMI

Any user-defined command files must also be modified accordingly and run before Rev. 19.4 is used.

SOFTWARE PROBLEMS FIXED

Multi-line FINDS using variables in procedures now give correct results (SPAR 2005470).

Table data in files are now displayed correctly (SPAR 3000730).

The RANGE command is now included in the HELP function (SPAR 3001130).

PROC DELETE now works on procedure names with more than six characters (SPAR 3001135).

Computing on dates of 1990 or later now works correctly (SPAR 3001604).

A CHANGE operation will now only execute on a 'Y' or a 'YES' response (SPAR 3001885).

Find < on character data now works properly (SPAR 3002004, 3001236).

DESTROY now leaves the PKFILE with the RW locks it had before the DESTROY (SPAR 3002258).

If the directory a file is stored in has DALURW ACL's without 'P', POWERPLUS now traps this as an error (SPAR 3002484).

Screens now allocate 5 spaces for INT*2 and 10 spaces for INT*4 (SPAR 3002721).

Text lines are now printed correctly (SPAR 3002723, 3003943).

DESTROY -DELETE now works correctly on all files (SPAR 3002762, 3006068).

HEADING PAGE now reads from within a procedure file, if desired (SPAR 3002772).

EXPAND command now works correctly on MIDAS files created with a field as ASCII (SPAR 3002774).

EDIT PROC on a procedure that contains a period now works correctly (SPAR 3002817).

DISPLAY USING FORMnn using linked files now formats the output correctly (SPAR 3003390, 3001123).

Headings in writing report no longer hang (SPAR 3003734).

A REPORT CHANGE no longer prints superfluous characters (SPAR 3003778).

DISPLAY USING FORMnn no longer prints the MIDASPLUS error message 'UNIT NOT OPEN' when PRINT ERROR is ON (SPAR 3003929, 3004060, 3000733).

Data from three linked files can now be displayed without POWER hanging (SPAR 3004059).

Extra lines are no longer generated in reports (SPAR 3004062).

REPORT CREAT is now successful when AUDIT is on (SPAR 3004387).

PRINT without REPORT option no longer acts as a display on a screen (SPAR 3004889).

Previously, the first record of a file showed on a FORM followed by blank forms. Now all records appear (SPAR 3005395).

Previously, using linked files and forms, the first record would appear followed by a blank form, then the second record followed by a blank form and so on. Now all records appear without blank forms (SPAR 3005396).

When trying to print descriptors instead of displaying them without having a report file, 'REPORT DOES NOT EXIST' appeared. Now PRINT works correctly (SPAR 3005397, 3006084).

Descriptors NUM5 or NUM6 with pictures of 9(10)V99 or greater, no longer print a space with extra digits following (SPAR 3006058).

Calculations can now be made and the results assigned to a variable (SPAR 3006384).

An illegal FIND will no longer bump the user into PRIMOS level (SPAR 3006497).

ADD USING SCREEN and ADD work correctly and consistently now (SPAR 3006549).

SORT will now sort over 32,767 records successfully (SPAR 3006618).

When multiple users access the same FORM simutaneously, POWERPLUS will no longer print the message 'FORM DOES NOT EXIST' (SPAR 3007039).

Documentation note: When doing a compound FIND, only one blank space can be left between the '#' and the rest of the command. If more than one space is, left an 'INVALID FIND' message will result (SPAR 3007148).

Report totals no longer double with link files (SPAR 3007260).

Member data descriptors are now displayed properly with the number of rows greater than one (SPAR 3007781).

The number of report lines written in a report are now printed correctly (SPAR 3007783).

Text data is now displayed on the row indicated (SPAR 3007784).

Data which is entered with negative exponents is now added and displayed correctly (SPAR 3009007).

Member file descriptors are now displayed when using the FORM function (SPAR 3009067).

DOCUMENTATION MODIFICATION

On page 7-14, Prime POWER Guide, add item 4. to the bottom of the page: 4. When doing a multi line FIND, do not space more than one space from the # prompt.

ENVIRONMENT

Rev. 19.4 PRIME POWER must be installed with Rev. 19.4 PRIMOS, FORTRAN, and MIDAS.

INSTALLATION AND BUILD PROCEDURES

It is essential that the install command file supplied in POWERPLUS be used to install Rev. 19.4 POWER. This command file runs the utilities to convert the procedure directory and the password file.

PRISAM

NEW FEATURES AND CHANGES

Sequential Files: PRISAM Rev. 2.0 now supports a new file organization, the sequential file. This new file organization is supported through the runtime library calls as well as by FAU and DIAG features.

Subsystem Access: Access to more than one ROAM based subsystem is now allowed from within a single runtime application and within a single runtime transaction. Specifically, access to DBMS and PRISAM is allowed to occur from within the same transaction.

Indexed File Internal Structure: Indexed file nodes now have an improved internal structure, which increases the performance of runtime calls such as Z$INSR, Z$UPDT, and Z$DELE. Users may expect to see a 10% to 20% performance increase for these calls over PRISAM Rev. 1. The new node structure also requires less disk space to contain the after-images of changed index nodes.

Note

Users of PRISAM Rev. 1 must convert their PRISAM files using the REV2_CONVERT utility before running PRISAM Rev. 2.0. Please refer to Appendix F of the PRISAM User's Guide for details.

Spanning of Partitions: PRISAM files may now span both logical and physical disks on a single network node. For all file organizations, the data component of the file may span more than one partition or UFD. For indexed organization, the index subfiles may be placed in partitions or UFDs other than that of the master. Users may specify up to 16 external partition or UFD names through the FAU.

Users may control the dynamic extension of the data component of indexed and sequential files by setting a maximum quota on those UFDs where the data component is to reside. For example, if three external data subfiles are specified at file creation time, and the UFDs that will contain them have their quotas set at 25, 50, and 0, respectively, PRISAM will insert data into UFD 1 until it exceeds 25 records (PRIMOS disk records, not PRISAM records), then start inserting into UFD 2 until it exceeds 50 records, then UFD 3 (which has no quota) until all records have been inserted or the disk becomes full. (Note that if the UFDs are located on separate partitions, then a disk full condition for the first two UFDs is equivalent to quota exceeded, and PRISAM will move to the next UFD when inserting data. Only when the last UFD becomes full does PRISAM report an error.)

For indexed files, users may increase the quota of a data UFD that has become full, thereby making the data component in that UFD available for more data once all other UFDs are full. If, in the above example, all UFDs are full, the user may increase UFD 2 quota from 50 to 100 (or delete unnecessary files in that UFD to lower the records used below 50). Since PRISAM cycles through each UFD looking for available space, it will see that UFD 2 is not full and resume inserting data there.

Time Limit: PRISAM now supports the new ROAM feature that enables an installation to specify a limit on the time that PRISAM waits for a concurrent read or write request to be satisfied. The system-wide default is set to 9 seconds. This may be overridden by using the CPRISAM utility to set the time limit to a user-specified value.

Note

The page wait feature may also cause timeouts which did not occur in PRISAM Rev. 1. This may occur if a file is being accessed by more than one user and a record that is locked by one user is requested by another. For more information, please refer to the PRISAM User's Guide.

If a PRISAM application is running in a mixed mode environment (that is, accessing DBMS and PRISAM files), then both PRISAM and DBMS will attempt to set the page wait timeout for ROAM. Whichever subsytem is invoked last sets this parameter. If it is PRISAM, page wait timeout is set to 9 seconds (and may be modified by using the CPRISAM utility). If the last subsystem to invoke ROAM is DBMS, page wait timeout is set to wait forever as is currently done in PRISAM Rev. 1.

DIAG DUMP Facility: The DIAG utility now provides a DUMP facility that displays the contents of data records in indexed, relative, and sequential file organizations and the contents of indexes within indexed files. The DUMP facility also displays the contents of the data description tables in Subfile 0.

FAU COPY Facility: The FAU utility now provides the COPY facility that enables the Administrator to copy PRISAM files through the FAU utility.

New Option: A new option has been added to the DIAG utility OUTPUT command. If the user specifies NO WAIT, DIAG will not prompt the user with —More— during command output displays that generate more than one screen full of information. This feature is useful for CPL programs that invoke DIAG and have had the problem of DIAG ignoring valid commands that happened to occur in the input stream in response to the —More— prompt.

The syntax for the wait option of the OUTPUT command is:

```
OUTPUT   { TO TTY [AND TO FILE pathname] }  [ WAIT      ]
         { TO pathname [AND TO TTY]      }  [ PAUSE     ]
                                            [ NO WAIT   ]
                                            [ NO PAUSE  ]
```

pathname is a simple filename or the pathname to a file.

For example, with the following command all command output will be directed to the user's terminal and there will be no prompt for —More— if the output of a command exceeds one screen.

```
OUTPUT TO TTY NO WAIT
```

As another example, as a result of the following command, DIAG will prompt the user if command output would exceed one screen. Since the output device was not specified, it will remain the same as in the preceding example (TTY), assuming this command follows the last one.

```
OUTPUT WAIT
```

Note

If output is directed to a file only, the wait_option has no effect (output to a file only is never paused).

This is an enhancement in response to SPAR 3009096.

New Routine: Z$INFO, a new user-callable runtime routine is now available. Z$INFO returns the last error code generated by a call to a PRISAM runtime routine and is very useful for COBOL users who get COBOL general error code 99. See the PRISAM User's Guide for more information.

SOFTWARE PROBLEMS FIXED

FAU UNLOAD of a file that contains integer, floating point, or packed decimal fields may cause records to split when using the COBOL or TEXT option. Data input files loaded or unloaded using the COBOL or TEXT option may not contain the ASCII code for the line feed character aligned on a byte boundary. FAU will read such a record as two records and attempt to load them as such. If the user has records containing

items of these data types, the user should unload and load them using the IMAGE or FINBIN options (SPAR 3010512).

There is no mention in the documentation of the limit on number of duplicate keys allowed or how the maximum number can be exceeded. The limit is 64k minus 2 duplicates and may be exceeded by performing many insert delete operations on records with duplicate keys (SPAR 3009203).

The documentation states that the range of tenths of seconds for open wait in PRISAM CONFIG is 0 to 32767. The correct range is 1 to 3600 (SPAR 3009363).

Z$KDEL returned ER$HOF when key was greater than any key in the file; ER$LOF when less than any key in the file. Z$KDEL now returns ER$NFD (SPAR 3007968).

The FAU global command USE (specifying a current file) could not be properly used without first performing another FAU function on that file. USE now functions properly (SPAR 3009667).

Using FAU CREATE on a UFD that does not exist caused FAU to crash with a pointer fault. CREATE now functions properly (SPAR 3010487).

FAU LOAD of a file with over 400,000 records resulted in a file with empty indexes (this is a duplicate of a PACK bug that was fixed in Rev. 1.0.1). LOAD now functions correctly (SPAR 3007143).

The error ER$FEX was missing from the SYSCOM user error insert files. It has been returned (SPAR 3006784).

An enhancement request to have a routine that returns the last error code is very useful for COBOL users who get COBOL general error code 99. The new routine is called Z$INFO. See the PRISAM User's Guide for more information (SPAR 3005701).

In the SYSCOM user error insert files, ER$LFR was incorrectly replaced by 228. The correct number is now 227 (note that this leaves a gap between ER$LFR(227) and ER$INK(229)) (SPAR 3011210). The following insert files were changed:

```
SYSCOM>PRISAM_USER_ERRS.INS.PL1
SYSCOM>PRISAM_USER_ERRS.INS.FTN
SYSCOM>PRISAM_USER_ERRS.INS.PMA
```

The user key mnemonic for Z$KYST, K$CHAR, has been changed in the SYSCOM files to K$CHTR to resolve a key conflict with another PRIME product (SPAR 3011211). The following insert files were changed:

```
SYSCOM>PRISAM_USER_KEYS.INS.PL1
SYSCOM>PRISAM_USER_KEYS.INS.FTN
SYSCOM>PRISAM_USER_KEYS.INS.PMA
```

Under certain multi-user, high-contention circumstances, a read or a find of a record could return a ER$NFD code when the record is actually present. The code ER$ABT is now returned (SPAR 3011212).

The PRISAM documentation did not describe the following EPF restriction on the use of static mode libraries (SPAR 3010975).

> Because the linkage areas for static mode libraries are all statically allocated, a user can have only one program EPF using the same static mode library at any given time.

For example, if a user resumes a program EPF that accesses PRISAM and the program also tries to resume another program that contains a CP$ call to invoke another PRISAM program, the second program will fail with an error similar to:

```
Error condition LINKAGE_ERROR raised at .....
Attempt to link to an in use static mode library at
entry point "nnnnnn".
```

DIAG 'HELP DUMP' information was incorrect and the terms RBODY and RBASE are replaced with SRCD and SRDB (SPAR 3009051, 3009013).

FAU LOAD of a relative file loaded spurious data if it was preceded by a partial load into the same relative file (SPAR 3009659).

If the user is in an update transaction with more than one PRISAM file open and becomes deadlocked with another user, ROAM aborts the transaction. PRISAM detects the transaction abort but causes the user's process to be aborted. There is no damage to any of the user's files when this occurs. Before restarting, the user must perform a manual CLUP. The error log indentifies this problem with a "File ID mismatch" message (SPAR 3009020).

If a user attempts to open a PRISAM Rev. 1 file with PRISAM Rev. 2.0, the condition ACCESS_VIOLATION$ is raised and the program aborts. FAU and DIAG do not check for software and file revision compatibilty (SPAR 3009196).

OUTSTANDING PROBLEMS

When two or more users attempt to PACK separate files in the same UFD simultaneously, FAU may fail with the error "file in use. (CMBN$S)". This happens when the PACK function attempts to open a temporary file and should be a rare occurrence, since the attempts to open the temporary files must occur at precisely the same time (SPAR 3010163).

FAU CREATE does not tell the user if an attempt is made to create more than sixteen external data subfiles. ROAM informs the user but only after it has created sixteen and attempts to create the seventeenth (SPAR 3009017).

FAU creates an external index subfile if the user is attached to a UFD different from that of the master file. This only occurs if the file is a multiple-keyed indexed file and there is an EXTSFF file which does not declare all the keys in KREF statements but has at least one declared. FAU will create an external index subfile in the current attach point that contains all those keys not declared. The workaround is to explicitly declare every KREF in the EXTSFF file or, if external index subfiles are not needed, make sure there are no KREF statements in the EXTSFF file (SPAR 3010440).

When an empty Rev. 1 file is converted to Rev. 2.0 format using the REV2_CONVERT utility, the index subfiles are not properly converted. The remedy is to simply create the Rev. 2.0 file using FAU rather than converting the empty Rev. 1 file to Rev. 2.0 (SPAR 3009795).

If the maximum quota on a directory is exceeded during the FAU LOAD operation, FAU will report 'disk full' instead of 'maximum quota exceeded'.

When there is a request to read a record beyond the boundaries of a file, COBOL translates the PRISAM error to 'not found' (SPAR 3007632).

If either of the conditions 'maximum quota exceeded' or 'disk full' occurs during creation of a PRISAM relative file, PRISAM makes an unnecessary entry in the error log (SPAR 3008366).

DIAG halts if a user attempts to dump more than 7 nodes consecutively in certain PRISAM indexed files (SPAR 3009052).

ENVIRONMENT

PRISAM Rev. 2.0 requires PRIMOS Rev. 19.4 and ROAM Rev. 19.4. No previous revisions of PRIMOS or ROAM will work correctly with Rev. 2.0 of PRISAM. PRISAM requires the use of shared segments 2204, 2205, 2207, and 2227, and of per-user static storage segment 6007, addresses from 50000 to 122777.

INSTALLATION AND BUILD PROCEDURES

The build is standard. Installation and general information follows.

Files on System Tape

PRISAM (UFD): The top level UFD contains the following files:

PRISAM.INITINSTALL.COMI	Used to install PRISAM for the first time
PRISAM.INSTALL.COMI	Used to install PRISAM at any other time
PRISAM.SHARE.COMI	Used to share and initialize the PRISAM shared segments.

PRISAM>CMDNCO: This UFD holds the CPL interludes to those V-mode segment directories found in PRISAM>SEGRUN* that become external commands to PRIMOS. The following interludes are moved to CMDNCO on the command partition by PRISAM.INSTALL.COMI.

DIAG.SAVE FAU.SAVE

PRISAM>INFO: This UFD contains the documentation for the current release of PRISAM, PRISAM.RUNI, and PRISAM.RUNO.

PRISAM>LIB: This UFD contains the PRISAM binary object libraries: PRISAMLIB.BIN (the shared PRISAM runtime library) and FAULIB.BIN (the unshared PRISAM utilities library).

PRISAM>PRISAM*: This UFD contains the PRISAM error message file, ERROR_TEXTS.ERR, segment directories used for configuring and initializing PRISAM, the Rev. 1 to Rev. 2.0 File Conversion Utility, and the HELP directory for the PRISAM File Administrator Utility, as follows:

ERROR_TEXTS.ERR REV_NUM

CPRISAM.SEG IPRISAM.SEG REV2_CONVERT.SEG SPRISAM.SEG

HELP

PRISAM>SEGRUN*: This UFD contains the segment directories for the commands in PRISAM>CMDNCO:

DIAG.SEG FAU.SEG

PRISAM>SYSCOM: This UFD contains language source files which may be included into user programs. These files define the entry points, user keys, and error codes for PRISAM. The installation procedure copies these files to SYSCOM on the command partition.

PRISAM_USER_DCLS.INS.PL1
PRISAM_USER_ERRS.INS.FTN
PRISAM_USER_ERRS.INS.PL1
PRISAM_USER_ERRS.INS.PMA
PRISAM_USER_KEYS.INS.FTN
PRISAM_USER_KEYS.INS.PL1
PRISAM_USER_KEYS.INS.PMA

PRISAM>SYSTEM: This UFD contains the PRISAM shared segment run files:

PR2204 PR2205 PR4000

Instructions For PRISAM Installation

1. Attach to the MFD where you wish to restore the PRISAM UFDs.
2. Restore the UFDs supplied on tape.
3. Execute the PRISAM install program. If this is the first installation of PRISAM, at the supervisor terminal, type:

   ```
   CO PRISAM>PRISAM.INITINSTALL.COMI
   ```

   If it is not the first installation, type:

   ```
   CO PRISAM>PRISAM.INSTALL.COMI
   ```

4. If the PRISAM* directory is to be an ACL directory, set the following specific ACLs: SYSTEM:ALL and <user or group name(s) representing all PRISAM users>: ALURW (or just simply $REST:ALURW). These rights are the minimum required for PRISAM users. Then set specific ACLs for each file installed into PRISAM* to: SYSTEM:ALL and $REST:LUR. These six files are mentioned above under PRISAM>PRISAM* and must not be modified by users.

   If the PRISAM* directory is to be a password directory (not recommended), it is safest to allow all users to be owners having at least RW rights. If PRISAM users must be non-owners, they must have at least RW rights.

5. Share PRISAM from the supervisor terminal:

```
CO SYSTEM>PRISAM.SHARE.COMI
```

This command should be included in PRIMOS.COMI (C_PRMO) in CMDNCO so that PRISAM will be shared any time the system is cold started.

6. Convert all PRISAM files built under PRISAM Rev. 1.

Note

All existing PRISAM files created and/or accessed under PRISAM Rev. 1 must be converted to the new Rev. 2 format before they are accessable under PRISAM Rev. 2. The utility PRISAM*>REV2_CONVERT must be run against each such file. Refer to Appendix F of the PRISAM User's Guide for details.

ROAM

NEW FEATURES AND CHANGES

Support for timed wait on transactional reads and writes has been added at Rev. 19.4. PRISAM also supports timed wait.

Manual CLUP security has been improved. A ROAM user can no longer CLUP another user unless the first user issues the CLUP command from the supervisor terminal or is a member of the .ROAM_ADMIN ACL group.

IROAM now checks to be sure if the date and time are set on the system before it initializes ROAM. If it is not set, the operator is instructed to set it and then rerun IROAM.

Support for mixed DBMS and PRISAM transactions has been implemented in this version of ROAM.

The number of available file units per ROAM process has been increased and ROAM is compatible with PRIMOS' dynamic file unit allocation at Rev. 19.4.

This version of ROAM uses the per process dynamic storage allocation mechanisms offered by Rev. 19.4 PRIMOS.

SOFTWARE PROBLEMS FIXED

Use of MAGSAV and MAGRST from the supervisor terminal is now allowed for DBMS/ROAM schemas. However, use of MAGRST from the supervisor terminal is not allowed if a version of the DBMS/ROAM schema already exists on the system. A user must be a member of the .DBMS_ADMIN group to delete a DBMS/ROAM schema and User 1, the supervisor terminal, can not be a member of any user profile group.

ROAM writes all after-image file messages to the system console even when terminal output is disabled (SPAR 3009353).

The start and end times of automatic clean up will be recorded in the ROAM system log whenever a user is force logged out of PRIMOS. If the end time is not written to the log, it means that automatic clean up did not have time to complete and manual clean up must be run. The format of the message is:

```
CLUP FORCELO 0 Status User-login-id User-# Date/time
```

The value of Status is 'S' for start of automatic cleanup and 'E' for end of automatic cleanup.

The ROSAU STATUS command now displays how full the active AIFILE is.

Access to ROAM files that have been truncated or altered by FIX_DISK is no longer allowed (file verification).

When a data file is extended during ROLLF and the file does not end on a page boundary, the system error reported has been clarified (ROAM system error handler).

ROAM now checks at open time to assure that the revision of the data file is compatible with the revision of the ROAM system that is running. Although the revision of the data file has not been changed, this check has been added in the event that a change in the ROAM file format is required in the future (ROAM file revision check).

When a SAVE of the AIFILE is done and an incomplete transaction is in the active AIFILE, SAVE will not lose that transaction by moving the next save address past its beginning (SPAR 3008558).

The ACLs for .ROAM_ADMIN are now set correctly after MAGRST (SPAR 3003079).

The CLUP command now asks the user to run IROAM if ROAM is not initialized (SPAR 3008116).

A change was made to allow MAGSAV of ROAM files from the current directory. This problem was never seen in a released version of ROAM but was introduced with the FIX_DISK change above and is included here for completeness (SPAR 3007397).

A system error is no longer reported when deleting a subfile from an archived ROAM file (SPAR 3007058).

The list of request mode for each user has been initialized before the user requests a general lock (SPAR 3005070).

When MAGSAV -INC is done, ROAM now checks the directory entry of the Master and each Slave to see whether or not the whole file has to be saved (SPAR 3006538).

An appropriate error message is now returned if an improper request id for a lock is not in use (SPAR 3005071).

Exceeding Disk/Quota space when expanding a DBMS schema table no longer causes a system error (SPAR 3003602).

When finding the next name in the RCVTAB, offline ROAM now verifies the position of the previous name in the RCVTAB (SPAR 3007249).

Offline ROAM returns a user error rather than a system error when the number of files open exceeds the maximum allowed (SPAR 3006061).

All file units are now closed after the initial installation of ROAM has been completed (SPAR 3009346).

A problem encountered during Rev. 19.4 testing that caused certain retrieval transactions to be treated as update transactions has been fixed and performance has thus improved. This problem was never seen in a released version of ROAM but was introduced with mixed mode transaction support and is included here for completeness (SPAR 3009083).

The improved EXIT$ condition signalling mechanism with program invocation level that PRIMOS provides is utilized by ROAM.

The new condition WARMSTART$ is ignored by the ROAM automatic cleanup mechanism.

OUTSTANDING PROBLEMS

Attempts to manipulate RBFs at the MFD level cause 'Illegal treename' errors as well as a system error when trying to delete an RBF (SPAR 3010476).

The system allowed the supervisor terminal user to enter ROSAU while the ROLLB phantom was running. This resulted in a system error and ROAM aborting (SPAR 3009683).

ROAM does not correctly determine whether a data base has been saved if the data base has been deleted, restored, and then had its name changed. This has only occurred when running from a CPL program (SPAR 3009470).

CLUP returns 'Not a valid Data Base Administrator' when a member of the ROAM_ADMIN group attempts to CLUP another user. However, any ROAM process can CLUP itself or another process with the same login id. Therefore, a user can run the CLUP command from any terminal to clean up the ROAM process by specifying the user number of that ROAM process (SPAR 3008863).

If users break (issue a <CTRL-P>) while BIFILE is being extended, the size of the Before Image file (BIFILE) as indicated by the BI block header is different from the actual physical BIFILE size (SPAR 3008029).

Attempts to improperly close the system file return a system error rather than a user error (SPAR 3008179).

ROAM returns an error when opening a segment directory if * has been specified as part of the pathname (SPAR 3007919).

ROAM conversion of a schema containing more than 128 subfiles fails with an adjust_schema (9) error (SPAR 3009058).

ROAM conversion fails if a subfile referenced in the schema table does not exist (SPAR 3009059).

PERMANENT RESTRICTIONS

Because the linkage areas for static mode libraries are all statistically allocated, a user can have only one program EPF using the same library at any given time. Therefore, ROAM could not be accessed by a single user from more than one EPF level at the same time. This is the EPF restriction on static mode libraries.

ENVIRONMENT

Rev. 19.4 ROAM requires PRIMOS Rev. 19.4. ROAM requires the use of shared segments 2200 through 2203, 2217, 2220, 2223, and 2224. ROAM also requires the use of private segments 6007 (words 0 through '47777) and 6011.

INSTALLATION AND BUILD PROCEDURES

The build is standard. Installation and general information follows.

Files on System Tape

ROAM (UFD): The top level UFD contains the following files:

ROAM.INITINSTALL.COMI	Used to install ROAM for the first time
ROAM.INSTALL.COMI	Used to install ROAM after its initial
ROAM.INSTALL.CPL	installation
ROAM.SHARE.COMI	Used to share the ROAM shared segments

ROAM>CMDNC0: This UFD holds the R-mode interludes to those V-mode segment directories found in SEGRUN* that become external commands to PRIMOS. These interludes are moved to the top level CMDNC0 by ROAM.INSTALL.COMI.

CLUP.SAVE	CN_RBF.SAVE	COPY_RBF.SAVE	DELETE_RBF.SAVE
IROAM.SAVE	LIST_RBF.SAVE	REST_RBF.SAVE	ROSAU.SAVE
SAVE_RBF.SAVE	SET_RBF.SAVE		

ROAM>INFO: This UFD contains the documentation for the current release of ROAM, ROAM.RUNI and ROAM.RUNO.

ROAM>LIB: This UFD contains the ROAM binary object libraries:

ROOFFLIB.BIN (the ROAM offline library) and
RORUNLIB.BIN (the ROAM runtime library).

ROAM>ROAM*: This UFD contains the following:

3 Files: ROAM.ERR, ROLIB.CPL, TAPE_PH.CPL
2 Segment Directories: ROLIB.SEG, TAPE_PH.SEG
3 Directories: HELP, TOOLS, USER

ROAM.ERR is the ROAM error message file. ROLIB.CPL and ROLIB.SEG are the CPL file and the V-mode segment directory for the ROLIB (Before-Image Recovery) phantom. TAPE_PH.CPL and TAPE_PH.SEG are the CPL file and the V-mode segment directory for the TAPE_PH phantom (used to automatically save the After-Image file).

The HELP sub-ufd contains ROAM help files.

ROAM>ROAM*>HELP: This UFD contains help files for ROAM interactive commands. At present, only ROSAU (the help file for the ROAM System Administrator Utility) exists. The sub-UFD ROSAU.UFD contains help files for all ROSAU subcommands.

ROAM>ROAM*>HELP>ROSAU.UFD: Each file in this UFD contains information corresponding to a ROSAU subcommand of the same name. For example, the file ALLOCATE contains information on the ROSAU ALLOCATE subcommand. The files are as follows:

ALLOCATE	AUTOSAVE	CLEAR	INSTALL
CREATE	FIX	HELP	QUIT
LOCK	MEND	MOVE	SAVE
RECON	RESET	ROLLF	
STATUS	UNLOCK	WAIT	

ROAM>ROAM*>TOOLS: The TOOLS sub-UFD contains ROAM tools for system analysts.

ROUTL.CPL	ROUTL.SEG	RORUNLIB.MAP
ROOFFLIB.MAP	TRACE_RO.CPL	TRACE_RO.SEG

ROUTL.CPL and ROUTL.SEG are the CPL file and V-mode segment directory for the ROUTL tool. The TRACE_RO.CPL and TRACE_RO.SEG are the CPL file and V-mode segment directory for the TRACE_RO tool. Both tools are used to dump or trace runtime ROAM. THE RORUNLIB.MAP and ROOFFLIB.MAP are ROAM runtime and offline shared library maps.

The USER sub-UFD contains the ROAM system log and is where any user system error logs are created. The ROAM system log is a record of ROAM events. The log is kept for two days. Yesterday's log is named OLD.ROAM.LOG and today's log is named ROAM.LOG.

The user system error logs are named in the format USER<##>-<time>-<date>.LOG and contain valuable information about the state of ROAM when a system error occurs. If you should get a system error, be sure to spool the USER log and save it for your PRIME analyst to examine.

ROAM>SEGRUN*: This UFD contains the V-mode segment directories corresponding to commands in ROAM>CMDNCO.

CLUP.SEG	CN_RBF.SEG	COPY_RBF.SEG	DELETE_RBF.SEG
IROAM.SEG	LIST_RBF.SEG	REST_RBF.SEG	ROSAU.SEG
SAVE_RBF.SEG	SET_RBF.SEG		

ROAM>SYSTEM: This UFD contains the ROAM shared segment run files. The RRxxxx files correspond to the ROAM Runtime shared segments and the ROxxxx files correspond to the ROAM Offline shared segments.

RO2203	RO4000	RR2200
RR2201	RR2220	RR4000

Instructions For Initial Installation Of ROAM

To install ROAM initially, do the following:

1. Restore the ROAM UFD supplied on the system tape.

2. To execute the ROAM initial installion program, issue the command:

```
CO ROAM>ROAM.INITINSTALL.COMI
```

   This program will create the top-level ROAM* UFD on the command device and copy the ROAM system sub-UFDs to the appropriate top level system sub-UFDs. If desired, the ROAM* UFD may be moved to a partition other than the command device.

3. Set the system date and time. Be sure the system date-time is set before sharing ROAM. The ROAM initialization program will not run and ROAM will not be initalized if the date-time is not set.

4. Share ROAM from the supervisor terminal with the following command:

```
CO SYSTEM>ROAM.SHARE.COMI
```

   Note that when the ROAM initialization program (IROAM) is run, error messages are printed. IROAM is responsible for restoring ROAM's configuration data, validating the ROAM system files, and initializing ROAM shared memory. Since this is the installation of ROAM, no configuration data nor system files exist; thus, these messages may be disregarded.

5. Invoke the ROAM System Administrator's Utility (ROSAU) INSTALL subcommand to install ROAM. This entails creating the system files (the Recovery Table, the Before-Image file, and the After-Image file) and setting the mode of after-imaging. This installation will implicitly initialize ROAM's configuration data. For each system file created, the user will be prompted for the UFD in which to place each system file and, in the case of the Before- and After-Image Files, the number of pages to initially allocate to each. Creating the After-Image file and setting the mode of after-imaging (manual or automatic saves) is optional depending on whether or not autosave will be used at your site. The QUIT subcommand returns the user to the PRIMOS command level when installation is complete. The following rules for placement of the system files should be followed.

   - In order to ensure proper after-image recovery, the After-Image file should be placed on a different disk partition than the user's data base files.

   - In order to guarantee at least one of either before- or after-image recovery in the event of a media failure, the Before-Image file should be placed on a different disk partition than the After-Image file. In order to increase disk I/O performance, the Before-Image file should be placed on a different disk drive than the After-Image file.

   - In order to optimize the reconstruction of the Recovery Table or the Before-Image file, if lost in a media failure, the Recovery Table should be placed on a different disk partition than the Before-Image file.

   A rough estimate of the initial size of the Before- and After-Image files can be obtained by adding the cumulative size

of the Before- and After-Image files for all of your data base files.

6. Reshare ROAM again with the following command:

```
CO SYSTEM>ROAM.SHARE.COMI
```

This time, no error messages should be printed. This command should be included into the coldstart file (PRMOS.COMI or C_PRMO) in CMDNCO so that ROAM will be shared at any time that the system is cold started.

Instructions for Re-installation of ROAM

To re-install ROAM, do the following:

1. Restore the ROAM UFD supplied on the system tape.
2. Execute the ROAM installation program with the command:

```
CO ROAM>ROAM.INSTALL.COMI
```

3. Set the system date and time. Be sure the system date-time is set before sharing ROAM. The ROAM initialization program will not run and ROAM will not be initalized if the date-time is not set.
4. Reshare ROAM with the command:

```
CO SYSTEM>ROAM.SHARE.COMI
```

CHAPTER 6

INFORMATION

INFORMATION

NEW FEATURES AND CHANGES

Prime INFORMATION, Release 5.5 runs on PRIMOS Rev. 19.4 rather than Revisions 19.1, 19.2, or 19.3. It contains all the functionality of Release 5.4 plus all the fixes of Releases 5.4.1 and some fixes subsequent to 5.4.1. Refer to the Release 5.4 INFORMATION.RUNI file or the 5.4 MRU (MRU5964-004) for details of new functionality for Release 5.4. Details of fixes for Releases 5.5 are described below.

SOFTWARE PROBLEMS FIXED

Problems Fixed in Release 5.4.1

An INFO/BASIC program using exponential notation such as 1.0E+12, would not compile properly if the number was followed by + or - signs. These signs would be considered as part of the exponent. Thus, tokens succeeding the + or - sign would not be allowed. At Release 5.4.1, the INFO/BASIC compiler recognizes this syntax correctly (SPAR 3006245).

If a HEADING option was not specified for a report, then the top margin was never printed on the first page. At Release 5.4.1, a top margin will now be produced whenever it is specified by the SETPTR command (SPAR 3006694).

When the 'K' in field 1 of the VOC entry 'TO' was changed to 'KEYWORD', a GETLIST XX TO 1 would not recognize the word 'TO' as a keyword. This has been fixed at Release 5.4.1 (SPAR 3006783).

At Release 5.4, if the REUSE function in an I-Type expression was passed a null string, an access violation occurred. This is fixed at Release 5.4.1 (SPAR 3007455).

Prior to Release 5.4.1, the INFO/BASIC PAGE statement generated linefeeds to align to the bottom of a page even when the report did not have a FOOTER. This is fixed at Release 5.4.1 (SPAR 3007456).

An INFO/BASIC report that had no top and bottom margins produced approximately 64 blank lines at the end of the report. This is fixed at Release 5.4.1. In addition, if the top and bottom margins are set to zero and no HEADING or FOOTING is specified, paging is suppressed (SPAR 3007720).

If a SAVE.LIST was done against a large select list, the list written to disk would end up larger than the actual number of keys selected. At Release 5.4.1, the number of keys saved on disk is identical to the number of keys selected (SPAR 3007849).

The NOEJECT option was never displayed by SETPTR even though it was set. At Release 5.4.1, this has been rectified so that NOEJECT is always displayed when it is set (SPAR 3008320).

When a user encountered a 'disk quota exceeded' or 'disk full error' message and was requested to release some space from another terminal, the terminal would appear to be hung if another user attempted to access the same group. This was caused by the group lock being held until space was released. At Release 5.4.1, the group lock is released during this request, allowing the other users to continue.

If a printer report was aligned on a page boundary and PRINTER OFF, PRINTER CLOSE were executed, an extra page was produced. At Release 5.4.1, no extra page is produced.

Problems Fixed in Release 5.5

Error message 'Var Not Assigned' from Perform. The error message 'Var not assigned' appeared when the the Menu processor was exited either using the 'Q' option after the Menu processor had been invoked via a login paragraph or after a control-P was typed. This is fixed at Release 5.5 (SPAR 3006034).

Prior to Release 5.5, executing the sequence of statements PRINTER ON, PRINTER CLOSE, PRINTER OFF caused the NOHEAD option to appear in the SETPTR characteristics. At Release 5.5, this has been fixed and the statements listed above no longer affect the characteristics established by a SETPTR or a TERM command (SPAR 3006035).

Prior to Release 5.5, the INFO/BASIC CLEAR statement would not clear those variables assigned a string literal of length greater than eight characters. This is fixed at Release 5.5 (SPAR 3006246).

The sequence HUSH ON, PTERM -ECHO DELAY, HUSH OFF caused extra linefeeds to be echoed (SPAR 3006483). The HUSH and PTERM commands have been modified and the following should be noted:

HUSH ON suppresses the echoing of all terminal input and suppresses all terminal output except output generated by errors when invoking a PRIMOS primitive.

If the user is in ECHO DELAY and enters HUSH ON mode, all standard terminal output, except PRIMOS subsystem error output, is suppressed. This is achieved by exiting ECHO DELAY mode while in HUSH ON mode. If the ECHO DELAY mode was previously in effect, then ECHO DELAY mode is restored when leaving HUSH ON mode.

If the user is in HUSH ON mode and attempts to enter ECHO DELAY mode, INFORMATION postpones the effects of this protocol until the user exits HUSH mode. ECHO DELAY mode is entered at that point.

The PTERM command and the subroutine !PTERM did not behave consistently when given the same parameters. The !PTERM subroutine has been improved so it functions according to the PTERM command except that -DISPLAY is not supported (SPAR 3006586).

The TERM command was destroying the banner established for the default printer. At Release 5.5, the default printer banner is not affected by the TERM command (SPAR 3006957).

The JOB command did not allow options to be passed to the utility that was being invoked by JOB. At Release 5.5, the command containing the utility's name and options may be enclosed in single quotes. These options are not passed to the JOB processor but rather to the job. For example, the command:

```
JOB BASIC BP TEST
```

invokes BASIC with the arguments BP TEST. That is, the command compiles the program TEST in file BP. The following command compiles TEST with the -XREF option in the file BP:

```
JOB 'BASIC BP TEST -XREF'
```

The JOB processor options -DISPLAY and -STATUS function as before (SPAR 3007478).

SPAR 3007867: On-line help for FORMAT and FANCY.FORMAT. The on-line help for the FORMAT and FANCY.FORMAT verbs refer to the -LPTR option, whereas they should refer to the LPTR option.

Pathnames in fields 2 and 3 of type F VOC entries are changed. If the pathname of a type 1 file contained the partition name, the editor produced various error messages. This problem has been fixed at Release 5.5. Partition names are supported in fields 2 and 3 of the VOC entry for users' files (SPAR 3008470).

If a file had a record length of 2048 characters and printer unit 0 was in use, the file would be damaged under cetain circumstances. This is fixed at Release 5.5 (SPAR 3009070).

ENVIRONMENT

Prime INFORMATION Release 5.5 requires PRIMOS Rev. 19.4.

INSTALLATION AND BUILD PROCEDURES

Attach to any MFD and restore the contents of the tape containing Prime INFORMATION Release 5.5. This creates a top-level UFD called INFORMATION.

Initial System Installation: The following steps should be followed to install Prime INFORMATION on a new system.

1. Run the INFORMATION.INITINSTALL.CPL File. Attach to INFORMATION and run INFORMATION.INITINSTALL.CPL. The CPL file determines if top-level UFD ISYS exists on MFD 0. (ISYS is Prime INFORMATION's master account.) If ISYS does not exist it is created. You should ensure that you have adequate access rights for this. INFORMATION.INITINSTALL installs the needed files in their proper places.

2. Create Catalog Segments. The INFORMATION Administrator has the ability to define the segments to be used for the catalog space. To do this, edit the file ISYS>SYS.CATALOG>SEGMENTINDEX to reflect the appropriate segments. The first number in the file signifies the number of segments that will be used. The subsequent numbers are the actual segment numbers. For example:

```
8
2330
2331
2332
2333
```

2334
2335
2336
2337

The eight segments (2330,2331,2332,2333,2334,2335,2336,2337) will be shared and used for the system catalog space. Note that these segment numbers need not be contiguous nor in sequential order.

The INFORMATION administrator should ensure that INFORMATION users have adequate access rights to ISYS and ISYS>SYS.CATALOG. Most installations give INFORMATION users LUR access to the ISYS UFD and LURW access to the ISYS>SYS.CATALOG UFD. Users who are permitted to reorganise the catalog space will need DALURW access to the ISYS>SYS.CATALOG UFD. User SYSTEM should be given LUR access to ISYS and DALURW access to ISYS>SYS.CATALOG.

3. Share the new system. From the supervisor terminal, attach to INFORMATION and run INFORMATION.INITSHARE.CPL. This CPL file will share the catalog segments that have just been defined, share the INFORMATION system, and run the INFORMATION bootstrap processor. The bootstrap processor will create a segment map for the catalog space and then load the new system into the catalog space. Once the file is run, re-attach to CMDNC0.

Note

The INFORMATION.INITSHARE.CPL file should only be run if you are installing a new system (that is, INFORMATION has never been installed on the machine on which you are doing the install).

4. Modify PRIMOS.COMI or C_PRMD. Add the command 'R SYSTEM>INFORMATION.SHARE.CPL' to the file PRIMOS.COMI (C_PRMD) in CMDNC0 in order to automatically share INFORMATION each time the system is booted. Note that when the system is being booted, the MFD containing ISYS should be added before INFORMATION is shared.

Installing an Update Release

If your system is running Prime INFORMATION Release 5.3.3 or earlier, refer to the update installation instructions of the Release 5.4 INFORMATION.RUNI file or the Release 5.4 MRU (MRU5964-004), as you will need to move the catalog space files. If your system is running Release 5.4, or 5.4.1, follow the instructions below. You should backup the system prior to installation.

1. Attach to any MFD and restore the contents of the tape containing Prime INFORMATION Release 5.5. This creates a top-level UFD called INFORMATION.

2. Install the new Release. Attach to INFORMATION and run the command file INFORMATION.INSTALL.CPL. This command file will install the new Release over the existing one.

   The INFORMATION administrator should ensure that INFORMATION users have adequate access rights to ISYS and ISYS>SYS.CATALOG. Most installations give INFORMATION users LUR access to the ISYS UFD and LURW access to the ISYS>SYS.CATALOG UFD. Users who are permitted to reorganise the catalog space need DALURW access to the ISYS>SYS.CATALOG UFD. User SYSTEM should be given LUR access to ISYS and DALURW access to ISYS>SYS.CATALOG.

3. Share the new Release. From the supervisor terminal, run SYSTEM>INFORMATION.SHARE.CPL. This CPL file shares the proper catalog segments and the INFORMATION run machine.

4. Run BOOT.INFO.CPL. Attach to ISYS and run the CPL file BOOT.INFO.CPL. This file loads the new verbs into the catalog space.

CHAPTER 7

FORMS AND FED

FORMS

NEW FEATURES AND CHANGES

The startup default device list for FED now includes the terminal type PT200. A new section has been added to the formname questionnaire that shows the valid sizes for the PT200 and allows the user to select one size. If the PT200 is not in the current device list, display of this section of the questionnaire is suppressed. If the PT200 is in the list and the user does not choose a size, then the default PT200 size of 80 by 48 is used. The FDL file produced by the CREATE command will contain FDL that specifies the size of the PT200 screen used in the creation of the form.

SOFTWARE PROBLEMS FIXED

User numbers greater than 99 are now handled correctly (SPAR 3003467).

FED

NEW FEATURES AND CHANGES

The startup default device list for FED now includes the terminal type PT200. A new section has been added to the formname questionnaire which shows the valid sizes for the PT200 and allows the user to select one. If the PT200 is not in the current device list, display of this section of the questionnaire is suppressed. If the PT200 is in the list and the user does not choose a size, then the default PT200 size of 80 by 48 is used. The FDL file produced by the CREATE command will contain FDL which specifies the size of the PT200 screen used in the creation of the form.

SOFTWARE PROBLEMS FIXED

While detailing fields in FED MODIFY, a stringrange error occurred when verifying an I/O field partially in view. This is now fixed (SPAR 3007076).

An access violation occurred when detailing fields in FED MODIFY and a new field was added between two existing fields and then verified. This is now fixed (SPAR 3007077).

While detailing fields partially in view in FED MODIFY, the field would be truncated to what was actually in view. This is now fixed (SPAR 3007078).

Terminal size limitations within the PRIMEWAY environment have been fixed (SPAR 3005941).

The -TCB option on the FED build now works in the same manner as on the FORMS build (SPAR 3005942).

CHAPTER 8

COMMUNICATIONS

DPTX

NEW FEATURES AND CHANGES

Warm Start handling

Terminal Emulation (PTDSC)

The PST100 and PT200 terminals have an extra mode called DSC mode. This mode is invoked by issuing the command PTDSC while on a system with DPTX running. This invokes the terminal interface program, PTDSC. Upon exiting PTDSC the terminal reverts to its normal mode and again functions as a normal PST100 or PT200.

PTDSC is a user program designed to interface with the DPTX/DSC package. PTDSC, along with the PST100 or PT200 terminal, provides emulation of an IBM 3277 Model 2 display station.

The terminal interface program is invoked by typing the command 'PTDSC'. PTDSC first indicates the version of PTDSC being used (for example, PTDSC Rev. 19.4), and then responds by requesting that the user enter the name of a valid port to the mainframe:

```
STATION NAME:
```

This port name was set up during configuration of DPTX. Following a successful attach to an emulation port, the line is placed in half duplex mode and PTDSC proceeds to check terminal status.

The PTDSC program may be started from a CPL program. The station name may be provided by a CPL &DATA statement. No other input will be accepted from a CPL program.

The first item checked by PTDSC is the identification number of the PROMS in the terminal. If the PROMS are not DSC mode (all PT200 terminals have DSC mode PROMS) or the terminal is not a PST100 or PT200, the user will be told that the terminal can not run PTDSC and PTDSC will be exited. Following proper initialization, the current copy of the Virtual Buffer is displayed on the screen, the terminal is placed in DSC mode, and normal 3277 operation may begin.

PST100/P200 Differences

PST100 Function Key Mapping: Below is a chart of PST100 Function keys and their equivalent IBM function keys.

Keys are assigned as follows:

KEY	IBM	Normal	Control	Shift	Ctrl-Shift
ENTER	ENTER	ENTER	$	ENTER	$
PF11	PF1	PF1	PF1	PF1	PF1
PF12	PF2	PF2	PF2	PF2	PF2
PF13	PF3	PF3	PF3	PF3	PF3
PF7	PF4	PF4	PF4	PF4	PF4
PF8	PF5	PF5	PF5	PF5	PF5
PF9	PF6	PF6	PF6	PF6	PF6
PF4	PF7	PF7	PF7	PF7	PF7
PF5	PF8	PF8	PF8	PF8	PF8
PF6	PF9	PF9	PF9	PF9	PF9
PF1	PF10	PF10	PF10	PF10	PF10
PF2	PF11	PF11	PF11	PF11	PF11
PF3	PF12	PF12	PF12	PF12	PF12
PF10	T.REQ	T.REQ	T.REQ	T.REQ	T.REQ
PF14	CLEAR	CLEAR	CLEAR	CLEAR	CLEAR
F1	PA1	PA1	PA1	PA1	PA1
F2	PA2	PA2	PA2	PA2	PA2
F3	PA3	PA3	PA3	PA3	PA3

Key	IBM	Normal	Control	Shift	Ctrl-Shift
F4	$	RECOV	RECOV	RECOV	RECOV
F5	$	EXIT	EXIT	EXIT	EXIT
F6	$	*	*	*	*
F7	$	$	$	$	$
F8	$	$	$	$	$

* Toggle keypad function between PF and Numeric. Keypad is initialized to PF function.

ERASE	$	ERASE (CHAR)	ERASE (INPUT)	ERASE (EOF)	ERASE (EOS)
STOP	$	CTRL P	CTRL P	CTRL P	CTRL P

Notes

All other escape sequences are NULL operations. $ means not implemented or NULL operation.

Three of the keys listed in the chart are not IBM function keys. These are:

Recover - writes a fresh copy of the virtual buffer to the screen.

Exit - causes an exit to PRIMOS. The terminal is put back into normal mode, PTDSC is exited, and the OK, prompt is displayed.

Stop - causes an exit to PRIMOS. The Exit key is the preferred means of exiting.

PST100 Discrepancies: All known discrepancies between PST100 3277 emulator and IBM 3277 are listed below.

- All PST100 character keys are typamatic.
- The only IBM indicator that is correctly emulated is INSERT MODE. The PST100's SYSTEM RDY indicator is similar to the IBM indicator SYSTEM AVAILABLE and the PST100 indicators KYBD LOCKED and SOFT LOCK are similar to the IBM indicator INPUT INHIBITED.
- The PST100 has no Field Mark key.
- The PST100 has no Dup key.

- The PST100 uses the ASCII character set. The IBM 3277 uses the EBCDIC character set. PTDSC provides the appropriate translation so the character set implementation is transparent to the user.

- The PST100 Reset key does not reset INSERT MODE.

- The PST100 Insert key toggles INSERT MODE.

- The PST100 displays null characters as dots.

- The PST100 Shift Lock key operates as a toggle. Also, the PST100 Shift Lock functionality affects alphabetic characters only. The IBM Shift Lock also shifts numeric and special character keys.

- When operating in INSERT MODE, the PST100 treats embedded nulls (nulls between characters) as regular characters and the editing extent is the current line or the current extent, whichever is more restrictive. The IBM 3277 will right-shift over embedded nulls and the editing extent is the entire screen.

- The editing extent of the PST100 Delete key is the current area and the line and screen will be wrapped as necessary. The editing extent of the IBM 3277 Delete key is the current line.

- The PST100 allows both '.' and ',' in numeric fields.

- The PST100 sets all MDT bits in unprotected attributes on an 'Erase Input' operation. This is done to help prevent the terminal and the Virtual Buffer Emulator from getting out of synchronization. The IBM Erase All Unprotected command is handled correctly. MDT bits are reset on execution of this command.

- The PST100 has an Erase Character key whereas the IBM does not. The Erase Character key deperssion will only erase the character at the current cursor position. Other characters in the field are not left shifted to fill the erasure as is done with the Delete key.

- The PST100 has an Erase to End of Screen key (actually, a key-combination) whereas the IBM does not.

- The PST100 has a HOME key. This is an IBM 3278 functionality, not a 3277 functionality.

PT200 Function Key Mapping: Below is a chart of PT200 Function keys and their equivalent IBM function keys.

Keys are assigned as follows:

KEY	IBM	Normal	Control	Shift	Ctrl-Shift
enter	ENTER	ENTER	$	ENTER	$
PF1	PF1	PF1	PF1	PF1	PF1
PF2	PF2	PF2	PF2	PF2	PF2
PF3	PF3	PF3	PF3	PF3	PF3
PF4	PF4	PF4	PF4	PF4	PF4
PF5	PF5	PF5	PF5	PF5	PF5
PF6	PF6	PF6	PF6	PF6	PF6
PF7	PF7	PF7	PF7	PF7	PF7
PF8	PF8	PF8	PF8	PF8	PF8
PF9	PF9	PF9	PF9	PF9	PF9
PF10	PF10	PF10	PF10	PF10	PF10
PF11	PF11	PF11	PF11	PF11	PF11
PF12	PF12	PF12	PF12	PF12	PF12
PA1	PA1	PA1	$	$	$
PA2	PA2	PA2	$	$	$
PA3	PA3	PA3	$	$	$
PA4	$	RECOV	$	$	$
F1	PA1	PA1	PA1	PA1	PA1
F2	PA2	PA2	PA2	PA2	PA2
F3	PA3	PA3	PA3	PA3	PA3
F4	$	RECOV	RECOV	RECOV	RECOV
F5	$	EXIT	EXIT	EXIT	EXIT
F6	$	$	$	$	$
F7	$	$	$	$	$
F8	$	$	$	$	$
F9	T.REQ	T.REQ	T.REQ	T.REQ	T.REQ
F10	CLEAR	CLEAR	CLEAR	CLEAR	CLEAR
ERASE	$	ERASE (CHAR)	ERASE (EOF)	ERASE (EOL)	ERASE (INPUT)
CHAR SET /STOP	$	CTRL P	CTRL P	CTRL P	CTRL P

Note

$ means not implemented or NULL operation.

PT200 Discrepancies: All known discrepancies between PT200 3277 emulator and IBM 3277 are listed below. Details regarding these differences may be obtained from the appropriate section of the Distributed Processing Terminal Executive Guide (DOC4035-193 and UPD4035-21A).

- All PT200 character keys are typamatic.
- The only IBM indicator which is correctly emulated is INSERT MODE. The PT200's SYSTEM RDY indicator is similar to the IBM indicator SYSTEM AVAILABLE, and the PT200 indicators KYBD LOCKED and SOFT LOCK are similar to the IBM indicator INPUT INHIBITED.
- The PT200 DUP key functionality is provided by the keypad key marked SCROL LOCK.
- The PT200 Field Mark functionality is provided by the unmarked keypad key between SCROL LOCK and ERASE.
- The PT200 key marked 'clear' (not to be confused with key F10 which performs the CLEAR function) performs the function of the IBM 'reset' key.
- The PT200 uses the ASCII character set, the IBM 3277 uses the EBCDIC character set. PTDSC provides the appropriate translation, so the character set implementation is transparent to the user.
- The PT200 Insert key toggles INSERT MODE.
- AID generating keys (ENTER, PF keys, PA keys, etc) reset insert mode. This is an IBM 3278 functionality, not a 3277 functionality.
- The PT200 Lock key operates as a toggle. Also, the Lock functionality affects alphabetic characters only. The IBM Shift Lock also shifts numeric and special character keys.
- The PT200 Shift/Lock key combination shifts all keys. This is compatible with IBM shift lock functionality. Also, the lock key alone causes exit from this mode.
- The PT200 allows both '.' and ',' in numeric fields.
- The PT200 sets all MDT bits in unprotected attributes on an 'Erase Input' operation. This is done to help prevent the terminal and the Virtual Buffer Emulator from getting out of synchronization. The IBM Erase All Unprotected command is handled correctly. MDT bits are reset on execution of this command.

- The PT200 has an Erase Character key, the IBM does not. The Erase Character key deperssion will only erase the character at the current cursor position. Other characters in the field are not left shifted to fill the erasure as is done with the Delete key.

- The PT200 has an Erase to End of Line key (actually, a key-combination) whereas the IBM does not.

Functional Differences Between the PST100 and PT200: The following differences exist between the PST 100 and the PT200.

- The PT200 supports Field Mark and DUP, the PST100 does not.

- The PT200 CLEAR (reset function) key resets insert mode, the PST100 RESET key does not.

- The PT200 displays NULLS as spaces, the PST100 displays them as dots.

- The PT200 has the capability of SHIFT LOCKing numeric and special characters whereas the PST100 does not.

- Insert mode on the PT200 is IBM compatible; there are minor incompatiblities on the PST100.

- Delete key functionality on the PT200 is IBM compatible; there are minor incompatibilities on the PST100.

- The PT200 has Erase to End of Line capability; the PST100 has Erase to End of Screen. Neither of these capabilities is found on IBM.

- The PT200 locks the keyboard itself on depression of a PF, PA, or F key. PTDSC performs this function for the PST100. Unimplemented keys on the PT200 will cause a momentary display of the SOFT LOCK indicator.

- The STOP key on the PT200 sends an XOFF. The PAUSE key on the PST100 sends an XOFF.

- The CHAR SET/STOP key combination on the PT200 sends a CTRL-P (exit to PRIMOS). The STOP key on the PST100 sends a CTRL-P.

Unusable Key Sequences: The following keys or key sequences should not be used during PTDSC operations. They are valid DPTX operations and are implemented to provide greater flexibility for user developed functions.

ESC ?	clears the screen without notifying PTDSC. The screen image gets out of phase with the Virtual Buffer image.
CTRL-RESET	(on PST100) same as ESC ?.
CTRL-CLEAR	(on PT200) same as ESC ?.
CTRL-CHAR SET	enter DSC mode. This causes a reinitialization of DSC mode without notifying PTDSC.
CTRL-SHIFT-CHAR SET	exit DSC mode. The terminal program leaves DSC mode without notifying PTDSC. The DSC sequences sent by PTDSC are now invalid.

QUIT Key Handling: The stop key or <CTRL>P can be used to exit to PRIMOS. To use either key, The terminal must be in on-line mode (indicated in the status line).

Error Messages

Listed below are error messages displayed by PTDSC, and their meaning:

THIS TERMINAL CAN NOT RUN PTDSC -

The terminal is not a PST100 or PT200 or, if it is, the wrong PROM set is installed.

EXIT PTDSC -

The exit key has been hit. A normal exit to PRIMOS is in progress.

TERMINAL IN RECOVERY MODE -

PTDSC is retransmitting a copy of the current screen to the terminal. This is due to the user hitting the RECOV key.

UNSOLICITED ACK

A T$IOK code was received from the BDI when one was not expected. This indicates possible software problems. After this message has been displayed for three seconds, the screen image is refreshed from the Virtual Buffer.

TRANSMISSION FAILURE

A T$TF code was received from the BDI. This may be caused by the IBM host issuing a select after a PF key was hit but before the terminal data was transmitted to the IBM host. It may also be caused by expiration of the DPTX NAK retry counter. After this message has been displayed for three seconds, the screen image is refreshed from the Virtual Buffer.

UNEXPECTED BAD RESPONSE

A T$TF code was received from the BDI when no response was expected. After this message has been displayed for three seconds, the screen image is refreshed from the Virtual Buffer.

TRANSACTION FAILED

A T$STAT code and T$TRSC subcode was received from the BDI. After this message has been displayed for three seconds, the screen image is refreshed from the Virtual Buffer.

UNKNOWN MESSAGE

An unrecognized code was received from the BDI. After this message has been displayed for three seconds, the screen image is refreshed from the Virtual Buffer.

UNABLE TO READ INPUT DATA, TRY AGAIN

PTDSC has made three unsuccessful attempts to read data from the terminal. This may indicate hardware problems in the link between the terminal and the PRIME host. After this message has been displayed for three seconds, the screen image is refreshed from the Virtual Buffer.

UNABLE TO SEND DATA, TRY AGAIN

The BDI has rejected terminal data. This may be caused by the screen image being out of phase with the Virtual Buffer. After this message has been displayed for three seconds, the screen image is refreshed from the Virtual Buffer.

ANY$ Condition: <Condition Name>

A PRIMOS condition has been signalled that PTDSC does not know how to handle. This is a fatal error. PTDSC will exit back to the PRIMOS command line. The condition name should be recorded for reporting to your PRIME Systems Analyst.

PRIMOS PROMPT (ER!) -

PTDSC has detected an error from which it cannot recover. It then puts the terminal back into normal mode and reverts to PRIMOS.

BD$xxx -

A fatal BDI error has occurred. Additional text will accompany these errors.

SOFTWARE PROBLEMS FIXED

Correction to system halt issuing a DPTX -OFF caused by page being unmapped during I/O.

Correction to PA2 key for terminal support.

When using DPTX subroutine BD$OUT, DPTX runs out of DB's on 88th record sent (SPAR 3006533).

In module BSCMAN, a resource block was not being released back to the free pool from BSCMTR to BSCMAN for a NAK in control mode (SPAR 3008133).

In module EAU, the line number that is used for CUTBL is computed incorrectly (SPAR 3007565).

EM1004

SOFTWARE PROBLEMS FIXED

The error message 'Queue length exceeded' will be generated whenever there is an attempt to add an entry to an RJQ table that already contains 400 entries. Because the limit on the size of the queue is 400, no queue entry number higher than this can be generated (SPAR 3002724).

RJOP: The LSITE command incorrectly displayed the list(s) of device(s) defined in the current SDF. This problem has now been fixed (SPAR 3005760).

When the user quits in response to the 'Do you wish to quit' question, input is taken from the terminal rather than a COMINPUT stream. If the user does not wish to quit, COMINPUT is resumed if it was active (SPAR 3001109).

EM200UT

SOFTWARE PROBLEMS FIXED

RJQ: The error message 'Queue length exceeded' will be generated whenever there is an attempt to add an entry to an RJQ table that already contains 400 entries. Because the limit on the size of the queue is 400, no queue entry number higher than this can be generated (SPAR 3002724).

The LSITE command incorrectly displayed the list(s) of device(s) defined in the current SDF. This problem has now been fixed (SPAR 3005760).

RJOP: When a user quits in response to the 'Do you wish to quit' question, input is taken from the terminal rather than a COMINPUT stream. If the user does not wish to quit, COMINPUT is resumed if it was active (SPAR 3001109).

EM7020

SOFTWARE PROBLEMS FIXED

RJQ: The error message 'Queue length exceeded' will be generated whenever there is an attempt to add an entry to an RJQ table that already contains 400 entries (SPAR 3002724).

This problem has been fixed by changing the file RJQ$ADD.PLP to set the error flag if the last entry in the queue was 400. The actual limit on the size of the queue has been set to 400; thus, no queue entry number higher than this can be generated.

The LSITE command incorrectly displayed the list(s) of device(s) defined in the current SDF. This problem has now been fixed (SPAR 3005760).

When the user quits in response to the 'Do you wish to quit' question, input is taken from the terminal rather than a COMINPUT stream. If the user does not wish to quit, COMINPUT is resumed if it was active (SPAR 3001109).

EMGRTS

SOFTWARE PROBLEMS FIXED

RJQ: The error message 'Queue length exceeded' will be generated whenever there is an attempt to add an entry to an RJQ table that already contains 400 entries. Because the limit on the size of the queue is 400, no queue entry number higher than this can be generated (SPAR 3002724).

RJOP: The LSITE command incorrectly displayed the list(s) of device(s) defined in the current SDF. This problem has now been fixed (SPAR 3005760).

The declaration of the transmit debug file data header has been changed so that the element 'emulator' is correct (SPAR 3006219).

RJOP: When a user quits in response to the 'Do you wish to quit' question, input is taken from the terminal rather than a COMINPUT stream. If the user does not wish to quit, COMINPUT is resumed if it was active (SPAR 3001109).

EMHASP

SOFTWARE PROBLEMS FIXED

RJQ: The error message 'Queue length exceeded' will be generated whenever there is an attempt to add an entry to an RJQ table that already contains 400 entries. Because the limit on the size of the queue is to 400, no queue entry number higher than this can be generated (SPAR 3002724).

RJOP: The LSITE command incorrectly displayed the list(s) of device(s) defined in the current SDF. This problem has now been fixed (SPAR 3005760).

The declaration of the transmit debug file data header has been changed so that the element 'emulator' is correct (SPAR 3006219).

RJOP: When a user quits in response to the 'Do you wish to quit' question, input is taken from the terminal rather than a COMINPUT stream. If user does not wish to quit, COMINPUT is resumed if it was active (SPAR 3001109).

EMX80

SOFTWARE PROBLEMS FIXED

The message 'JE15 Max number of characters exceeded' is displayed in the RJOP on transmission of a file.

The pointer into the output buffer will not get out of step with the virtual block size. The correct number of records which can be placed within a block are now correctly fitted into the block (Spar 3002303).

CONCAT: The message 'Illegal character in input file' will no longer be displayed within the RJOP.

Each sub-section within a CONCAT file is now processed correctly. Each section within a CONCAT file contains a header used to determine whether the sub-section contains text or binary records (SPAR 3006221).

The message 'Illegal character in receive file' no longer appears in the RJOP on transmission of binary files. The correct flag within the receive data structure is set on transmission of binary files (SPAR 3006220).

The message 'Invalid field address' is no longer displayed within the RJOP. Should a line within a card-image file contain exactly 81 characters, the correct message 'Line too long in transmit file' is displayed (SPAR 3006229).

On attempting to transmit transparent data from a master to a slave site, the line was dropped and the message 'Receive timeout - retries exceeded' was displayed in the RJOP.

If the SDF indicates that 3780 transparent data is to be sent, the flag indicating which protocol is in use will be set correctly.

The module TX80_TRAN.PLP has been modified to ensure that, on transmitting transparent data on a card-punch device, the initial block that defines the component selection contains the correct framing characters. The symptoms of this problem were that data could not be transmitted if the SDF indicated that data was to be sent in transparent mode (SPAR 3002946).

The error message 'Bad BCB received' was printed within the RJOP.

The RJE Ring 0 code has been changed so that the control message DLE/EOT is now added to the end of the data queue after any data so far received. This ensures that, when the line is subsequently disconnected, no data is lost. The worker module has also been changed to handle the control message and process it correctly. The modules RJES>RJEVNT.PLP, RJXMIT.PLP, and RJKEYS.INS.PLP have been changed to fix this problem within the RJE Ring 0 code. The modules WORKER.PLP and RX_TRANSLATE.PLP have also been changed (SPAR 3001095).

RJQ: The error message 'Queue length exceeded' will be generated whenever there is an attempt to add an entry to an RJQ table that already contains 400 entries.

This problem has been fixed by changing the file RJQ$ADD.PLP to set the error flag if the last entry in the queue was 400. The actual limit on the size of the queue has been set to 400; thus, no queue entry number higher than this can be generated (SPAR 3002724).

Reprocessing: The routines RX_TRANSLATE.PLP and RX80_TRAN.PLP have been modified to ensure that the device for which a save file has been generated is used to REPROCESS the file correctly (SPAR 3006636).

RJOP: The LSITE command incorrectly displayed the list(s) of device(s) defined in the current SDF. This problem has now been fixed (SPAR 3005760).

The problem in which the component selection for punch files was being written to the output file has now been fixed by a change to RX80_TRAN.PLP (SPAR 3005144).

When data is being received by a slave site, the default component selection has now been changed so that, when both a card-punch and a line-printer have been configured, the site will default to processing the first file received without component selection as a print file (SPAR 3006362).

The first file sent to a print device now has the correct setting of the associated parameters (for example, line length, forms control setting(s) and so on) (SPAR 3002535).

This problem occurs when attempting to transmit transparent binary data between the host and remote sites to a card-punch device. The routine TX80_BIN.PLP has been changed so that the last record of the file transmitted is not lost (SPAR 3003764).

The routine TX_TRANSLATE.PLP has been modified to ensure that all blocks queued to the RJI by the X80 worker are written to the transmit debug file (SPAR 3006205).

The declaration of the transmit debug file data header has been changed so that the element 'emulator' is correct (SPAR 3006219).

The error reporting within TX_TRANSLATE.PLP has been improved. Also, the initialization for transmit devices now occurs only once and only on the first block of the file (SPAR 3006228).

The declaration of fencht and fenchr in the frame_chars structure has now been corrected so that, when passed to TPUTL, their declaration agrees with that expected by this routine (SPAR 3006226).

CONCAT: The routine TX80_INIT.PLP has been changed so that the initialization of the flag TC.CONCAT is now handled correctly. This flag is used to indicate that the type of file queued is a CONCAT file (SPAR 3002114, related SPAR 3001502).

CONCAT: The routine TX80_INIT.PLP has been changed so that the initialization of the flag TC.CONCAT is now handled correctly. This flag is used to indicate that the type of file queued is a CONCAT file (SPAR 3001502, related SPAR 3002114).

RJOP: Fixes to modules INIT.PLP and CATCHALL.PLP so that when a user quits in response to the 'Do you wish to quit' question, input is taken from the terminal rather than a COMINPUT stream. If user does not wish to quit, COMINPUT is resumed if it was active (SPAR 3001109).

EMXBM

SOFTWARE PROBLEMS FIXED

The error message 'Queue length exceeded' was generated whenever there was an attempt to add an entry to an RJQ table that already contained 400 entries.

This problem has been fixed by changing the file RJQ$ADD.PLP to set the error flag if the last entry in the queue is 400. The actual limit on the size of the queue has been set to 400 so that no queue entry number higher than this can be generated (SPAR 3002724).

The LSITE command incorrectly displayed the list(s) of device(s) defined in the current SDF. This problem has now been fixed (SPAR 3005760).

The declaration of the transmit debug file data header has been changed so that the element 'emulator' is correct (SPAR 3006219).

RJOP: When a user quits in response to the 'Do you wish to quit' question, input is taken from the terminal rather than a COMINPUT stream. If user does not wish to quit, COMINPUT is resumed if it was active (SPAR 3001109).

PRIME/SNA Interactive Subsystem

NEW FEATURES AND CHANGES

The PRIME/SNA Interactive Subsystem is the member of PRIME/SNA that provides, in conjunction with the facilities available from the PRIME/SNA Server and PRIME/SDLC, emulation of a subset of the capabilities of the IBM 3270 Information Display System operating in an SNA environment. The PRIME/SNA Interactive Subsystem is composed of four parts:

- An LU Manager
- Terminal emulation
- Printer emulation
- An administration facility

The specific 3270 emulation provided by The PRIME/SNA Interactive Subsystem is a subset of the functionality available in the 3274 Control Unit, Models 1C and 51C, the 3278 Display Station, and the 3287 and 3289 printers.

PRIME/SNA Server Subsystem

NEW FEATURES AND CHANGES

The PRIME/SNA Server Subsystem is the member of the PRIME/SNA product family that, in conjunction with PRIME/SDLC, allows PRIME 50 Series machines to connect into SNA networks as a peripheral node of the network.

In addition to providing the SNA connection support, the PRIME/SNA Server Subsystem also provides a high level interface that allows PRIME 50 Series processes (in PRIME/SNA terms, LU Subsystems) to communicate with applications (in SNA terms, end users) existing within the SNA network.

There are two functional parts of the PRIME/SNA Server support:

- The PRIME/SNA Server Adminstration Support which is invoked through the PRIMOS external commands, SNA_SERVER and SNA_SERVER_CONFIG
- The PRIME/SNA Server process which is initiated as a phantom through the PRIMOS external command, SNA_SERVER