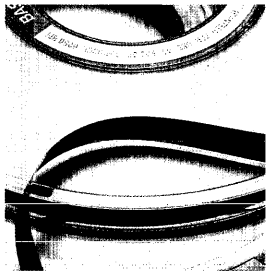
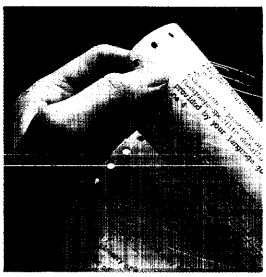
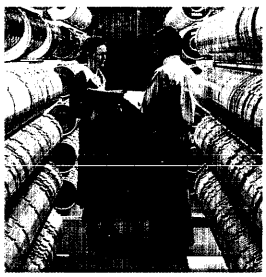
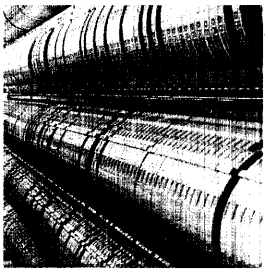
Prime Computer, Inc.

MRU4304-010P
Software Release Document
Revision 19.2

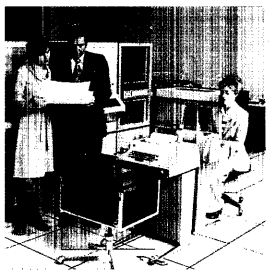
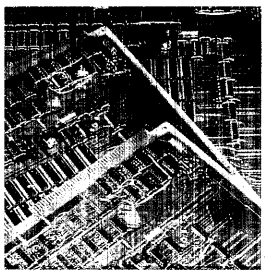










Software Release Document

MRU4304-010

Revision 19.2

by

Sarah Lamb and John Seybold

This guide documents the software operation of the Prime Computer and its supporting systems and utilities as implemented at Master Disk Revision Level 19.2 (Rev. 19.2).

Prime Computer, Inc.
500 Old Connecticut Path
Framingham, Massachusetts 01701

COPYRIGHT INFORMATION

HOW TO ORDER TECHNICAL DOCUMENTS

U.S. Customers

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

Prime Employees

Communications Services
MS 15-13, Prime Park
Natick, MA 01760
(617) 655-8000 X4837

Customers Outside U.S.

Contact your local Prime
subsidiary or distributor.

PRINTING HISTORY — Software Release Document

Edition	Date	Number	Documents Rev.
First	June 1983	MRU4304-010	19.2

SUGGESTION BOX

All correspondence on suggested changes to this document should be directed to:

Sarah Lamb
Technical Publications Department
Prime Computer, Inc.
500 Old Connecticut Path
Framingham, Massachusetts 01701

Contents

About This Book

This book summarizes the changes and new features in Prime's user software at Rev. 19.2 of PRIMOS. One chapter is devoted to each of the following:

- PRIMOS and Utilities (Chapter 2)
- Languages (Chapter 3)
- Data Management Systems (Chapter 4)
- Hardware Support (Chapter 5)

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 in the software at Rev. 19.2
- Documentation corrections and additions
- Software problems fixed at Rev. 19.2
- Software problems outstanding as of Rev. 19.2

Upgrade Paths to Rev. 19.2

The most common upgrade path to Rev. 19.2 is from Rev. 19.1. Rev. 19.2 introduces a number of new features and corrections to software problems. In addition, Rev. 19.2 incorporates all of the features, corrections, and changes that were introduced at Rev. 19.1.

New Features

This book describes features that are new at Rev. 19.2. For a summary of Rev. 19.1, refer to the Rev. 19.1 Software Release Document (MRU4304-009).

Documentation Corrections

This book contains a number of corrections and additions for 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 called NEW BOOK TITLES, in Chapter 1, for new titles at Rev. 19.2
- The Rev. 19.1 Software Release Document (MRU4304-009), for new titles at Rev. 19.1

This book does not repeat documentation corrections that were printed in previous Software Release Documents.

Software Problems Fixed

The corrected software problems listed in this book are those first fixed at Rev. 19.2.

Software Problems Outstanding

This book lists some of the software problems outstanding as of Rev. 19.2. For further information, contact your Prime field engineer.

CHAPTER 1

INTRODUCTION

IN THIS CHAPTER

This chapter provides:

- An overview of Rev. 19.2.
- Information on installation of Rev. 19.2.
- A list of new titles from Prime's Technical Publications Department.

OVERVIEW OF REV. 19.2

New Features and Products

Rev. 19.2 introduces the following new software products:

- Software support for a new Central Processor, the 9950
- Software support for the ICS2 Communications Controller

In addition, at Rev. 19.2, new features or enhancements have been added to the following products:

BOOT_CREATE
COPY_DISK
CPL
EDIT_PROFILE
EMACS
FIX_DISK
FORTRAN 77
LABEL
LD
LOGPRT
MAGLIB
MAGNET
MAGSAV/MAGRST
MAKE
MIDASPLUS
Pascal
PHYSAV/PHYRST
PMA
PRIMOS
PRINT_SYSLOG/PRINT_NETLOG
RFTNLIB
SEG
SLIST
Source Level Debugger
VFTNLIB
VRPG

Documentation Changes

This book contains documentation corrections or additions for manuals on the following products:

BASIC/VM
COPY_DISK
DBMS
DBMS/QUERY
EMACS
FORTRAN 77
MAGNET
MIDAS
MIDASPLUS
Pascal
PMA
POWERPLUS
PRIMOS
RUNOFF
SEG
Subroutines

Software Problems Fixed

Problems in the following products have been corrected:

BASIC/VM
COBOL
COPY_DISK
CPL
DBMS
DBMS/QUERY
FORTRAN
FORTRAN 77
LD
MAGLIB
MAGSAV/MAGRST
MIDAS
MIDASPLUS
Pascal
PHYSAV/PHYRST
PL/1, Subset G
PMA
EditorPRIMOS
EDIT_PROFILESEG
SORT
SPOOL
VFTNLIB
VRPG

Software Problems Outstanding

This book describes software problems outstanding in the following products:

COBOL
COPY
DBMS
DBMS/QUERY
MIDAS
MIDASPLUS
PL/1, Subset G
PMA
SEG

INSTALLING REV. 19.2

The Rev. 19.2 Master Disk is in complete Master Disk format, rather than in the update format that is normally used at minor (dot) revisions. The Rev. 19.2 Master Disk contains all Master Disk software, including the software that has not changed since Rev. 19.0. Rev. 19.2 can therefore be installed without prior installation of Rev. 19.0.

For information on installing a Rev. 19 system, refer to the Rev. 19.0 Planning and Installation Guide (DOC6426-190).

NEW BOOK TITLES

The following new technical publications are available at Rev. 19.2.

Primos and Utilities

DOC3060-192P System Architecture Guide [perfect bound]
DOC3060-192L System Architecture Guide [loose-leaf]

This book replaces the System Architecture Reference Guide (PDR3060-182). Intended for technical evaluators, system administrators, system programmers and planners, the new System Architecture Guide describes the internal functioning of Prime 50 Series and 9950 computers. It discusses process exchange, memory management, procedure calling, protection rings, hardware integrity mechanisms, interrupts, traps, checks, faults, and input/output. The new book also provides a complete listing of the instruction set for the 9950 and all Prime 50 Series systems.

DOC5029-192P Site Preparation Guide [perfect bound]
DOC5029-192L Site Preparation Guide [loose-leaf]

This guide is written for customers who are planning to install Prime equipment in their facilities. It describes activities typically associated with site preparation, from initial consideration through final checkout. It is addressed both to customers using Prime Customer Service and to customers using their own personnel for installation of Prime equipment in a new or existing facility.

UPD5037-192 System Administrator's Guide Update, Rev. 19.2

This update package describes new features affecting system administrators at Rev. 19.2, including changes for the ICS2 controller, changes to EDIT_PROFILE, and changes to shared libraries.

DOC7323-192P System Operator's Guide, Volume I [perfect bound]
DOC7323-192L System Operator's Guide, Volume I [loose-leaf]

This two-volume book provides guidelines for the successful daily operation of a Prime computer. Volume I is primarily concerned with hardware operations, such as system startup and shutdown. It also provides an overview of system organization and operation and explains operation of the Prime 9950, the Prime 2250, the 50 Series systems, including the 250-II, 550-II, 750, 250, 450, 550, 650, 850, and the 400 family, including the 350, 400, and 500.

DOC7324-192P System Operator's Guide, Volume II [perfect bound]
DOC7324-192L System Operator's Guide, Volume II [loose-leaf]

This volume covers system maintenance tasks, such as monitoring system resources and performing backups and restorations. It provides guidelines for performing these tasks and describes the utilities used to accomplish them. After an overview of PRIMOS and the operator's tasks, it details the general procedures an operator uses to keep the system running smoothly. Volume II also contains reference material on system operator commands and additional reference material on such topics as physical device numbers and error messages.

Languages

DOC3524-192P SEG and LOAD Reference Guide [perfect bound]
DOC3524-192L SEG and LOAD Reference Guide [loose-leaf]

This book is a complete revision of the LOAD and SEG Reference Guide (PDR3524-172). It should be used by anyone with Rev. 18.0 or later PRIMOS. The book includes an illustrated discussion of what happens on a default load, detailed steps for several types of advanced loads, and numerous new examples and sample programs.

DOC4029-192L FORTRAN 77 Reference Guide [loose-leaf]
DOC4029-192P FORTRAN 77 Reference Guide [perfect bound]

This book has been revised to include the REAL*16 quadruple-precision floating-point data type and the new functions that use it at Rev. 19.2 and later. The reader is expected to be familiar with some version of FORTRAN and with programming in general, but not necessarily with Prime computers. The book describes FORTRAN as a language, Prime extensions to F77, and F77 as it is implemented on Prime computers. It also explains how to compile, load, and run F77 programs and includes suggestions for the optimization of F77 programs.

UPD4303-192 Pascal Reference Guide Update, Rev. 19.2

This document updates the Second Edition of the Pascal Reference Guide (DOC4303-191) and covers the new STRING data type and other Pascal features new at Rev. 19.2. The STRING data type makes it easy to manipulate strings in Prime Pascal, as does PL/1G's CHARACTER VARYING type. A concatenation operator and nine built-in functions were also added at 19.2 to support STRING operations. This update makes several documentation corrections as well.

PTU2600-104 Rev. 19.2 Assembly Language Programmer's Guide

This Prime Technical Update describes the changes made to Prime's assembly language for Rev. 19.2. It covers the new quadruple-precision floating-point data type and the instructions that manipulate it. The update also contains new instructions for the Prime 850 and numerous corrections to previous documentation.

Data Management

DOC6291-192P DBMS User's Guide [perfect bound]
DOC6291-192L DBMS User's Guide [loose-leaf]

This book is a user's guide for DBMS, Prime's CODASYL-based Data Base Management System. It presents an overview of basic data base concepts, as well as general information about data base design, installation, programming, and recovery. The book focuses on concepts rather than software; other Prime documentation discusses DBMS software in detail. This book presumes that the reader has little or no familiarity with DBMS, but is familiar with Prime's operating system, editing facilities, and either COBOL or FORTRAN on Prime systems.

EMACS

DOC7446-192P EMACS Standard User Interface Guide [perfect bound]
DOC7446-192L EMACS Standard User Interface Guide [loose-leaf]

This book explains how to use the Standard User Interface (SUI). The SUI is an EMACS-based screen editor that runs on Prime's PST 100 and PT45 terminals. With the SUI, users can edit files using the terminal's function keys, which greatly simplifies all editing tasks.

Prime also supplies templates that fit over the function keys of the PST 100 and PT45 terminals, labelling the keys by function. These reduce the amount of memorization needed to use the Standard User Interface.

PTU2600-105 EMACS Rev. 19.2

This document describes the changes made to EMACS for Rev. 19.2.

Prime INFORMATION

DOC3909-053P Prime INFORMATION Primer [perfect bound]
DOC3909-053L Prime INFORMATION Primer [loose-leaf]

This primer is an introduction to Prime INFORMATION, an all-purpose data management system that enables both programmers and non-programmers to turn data into useful information. Topics covered include introductory concepts, file creation, file management, report generation, and an introduction to menus. The book is mainly tutorial and assumes little or no familiarity with data management systems. Several chapters contain advanced reference information to help users who have become more familiar with the system. An automobile dealership application is used as an example throughout the primer.

Office Automation

DOC6756-030P OAS Word Processing Guide (PT65) [perfect bound]
DOC6756-030L OAS Word Processing Guide (PT65) [loose-leaf]
DOC6757-030P OAS Word Processing Guide (PT45) [perfect bound]
DOC6757-030L OAS Word Processing Guide (PT45) [loose-leaf]
DOC6877-030P OAS Word Processing Guide (PST 100) [perfect bound]
DOC6877-030L OAS Word Processing Guide (PST 100) [loose-leaf]

Intended for the beginning OAS user. No background in programming, computer operations, or word processing is required. Knowledge of the Management Communications and Support module of Prime's Office Automation System is also not necessary.

There are separate books for each type of terminal--PT65, PT45, and PST 100. Each book gives complete instructions for using the Word Processing module of Prime's OAS. Word processing functions include creating, editing, and printing documents, as well as list processing to create personalized form letters and special reports. In addition to tutorial material, this guide includes an alphabetical reference section summarizing all editing functions.

A useful follow-up to this book is the OAS Management Communications and Support Guide.

DOC6755-030P OAS Management Communications and Support Guide [perfect bound]
DOC6755-030L OAS Management Communications and Support Guide [loose-leaf]

Intended for the beginning OAS user. No background in programming or computer operations is required.

This guide provides complete instructions for using all functions of the Management Communications and Support module of Prime's OAS on all terminals—PT65, PT45, and PST 100. These functions include sending, receiving, and filing electronic mail, electronically arranging appointments, and maintaining an automated calender.

Familiarity with the appropriate OAS Word Processing Guide may be helpful in using some of the MCS functions.

DOC6754-030P OAS Advanced Text Management Guide [perfect bound]
DOC6754-030L OAS Advanced Text Management Guide [loose-leaf]

Intended for the experienced Word Processing user whose system includes the Advanced Text Management module of Prime's OAS. Familiarity with the appropriate OAS Word Processing Guide is required.

This guide provides instructions for using all functions of Advanced Text Management. These include proofreading English documents, maintaining English and foreign language dictionaries, translating words from and into selected foreign languages, and searching documents for selected phrases. The book is written for users of all terminals—PT65, PT45, and PST 100.

DOC6781-030P OAS System Administrator's Guide [perfect bound]
DOC6781-030L OAS System Administrator's Guide [loose-leaf]

Intended for the OAS system administrator who is familiar with all OAS functions. No background in computer programming or operations is required. Familiarity with the appropriate OAS Word Processing Guide, the OAS Management Communications and Support Guide, and the OAS Advanced Text Management Guide is required.

This guide provides information on all System Administrator functions for OAS. These include creating and maintaining the system calender, network directory, user records, and document data base. Instruction is also provided for monitoring system usage, troubleshooting printer problems, and performing other tasks related to the administration of an OAS.

PTU2600-100 Using OAS on the PT25 Workstation

This update discusses the PT25 keyboard and the use of its function keys in OAS. Users can combine this document with the OAS Management Communications and Support Guide and with the OAS Word Processing Guide (PT45 or PST 100) to obtain a full set of instructions for using the PT25 with OAS.

PTU2600-102 Using OAS on Hard-copy and Non-OAS Standard Terminals

OAS can be used in teletypewriter (TTY) mode on hard-copy terminals and on video display terminals that do not meet minimum standards for OAS support. This update provides instructions for using TTY mode.

CHAPTER 2

PRIMOS AND UTILITIES

PRIMOS

NEW FEATURES AND CHANGES

> WARNING
>
> User software that uses any of the three reserved words in a directory entry must be changed. At Rev. 19.2, these words are used by PRIMOS.

Before converting to Rev. 19.2, run FIX_DISK to be sure these words are cleared. If you do not do so, some file utilities may fail with the message:

Improper access of restricted file

or LD may display a file attribute as "invalid".

Conversion Procedure for EDIT_PROFILE

System Administration Directories (SADs) created with Rev. 19.0 EDIT_PROFILE must be rebuilt before they can be read by 19.2 or later versions of EDIT_PROFILE.

Note

The SAD must be rebuilt before users log in under Rev. 19.2.

The SAD has to be rebuilt for the following reasons:

1. The 19.0 version of EDIT_PROFILE did not set the version number in the project files correctly. Although the 19.0 version runs under Revs 19.0 and 19.1, it fails at Rev. 19.2 with the message:

   Can't read project project id: bad version number.

2. If you have never used the Rev. 19.0 version of EDIT_PROFILE, you will not run into problem 1. However, you must rebuild your SAD in order to take advantage of the new features of EDIT_PROFILE at Rev. 19.2.

The System Administrator should rebuild the SAD project by project with the EDIT_PROFILE subcommand:

REBUILD -PROJECT project id

To do this, use EDIT_PROFILE Rev. 19.1. Execute the program EDIT_PROFILE.19.1.SAVE.

If you do not rebuild the SAD project by project, your other choice is to delete your old SAD and create a completely new one with the Rev. 19.2 version of EDIT_PROFILE.

For further details, see the EDIT_PROFILE section later in this chapter.

Number of Segments Available in PRIMOS

The limit on the total number of segments available for all users has been increased from 1022 to 8192.

Maximum Size of a Paging Surface

The maximum size of a paging surface is now 469278 records. For details, see the System Operator's Guide for Rev. 19.2.

Changes for New Processor

Some changes have been made to PRIMOS to support the 9950, a new processor introduced at Rev. 19.2. These include changes for the battery clock and enviromental sensors included with the processor. For details on these changes, see Chapter 5.

SETIME and MAXUSR Commands

The SETIME command now sets only system time and date, while its function of controlling user login after system startup is assumed by the MAXUSR command. The first MAXUSR command, issued after the system time is set, allows users to log in. A MAXUSR command issued before system time is set has no effect.

Larger Physical Memory

PRIMOS now supports physical memory of up to 16 megabytes for those processors that support it in the hardware. The previous limit was 8 megabytes.

Command Processor

The following selection criteria have been added to the wildcard portion of the command line processor:

-MODIFIED_AFTER date -MDA date	Same as -AFTER option (added for uniformity of user interface)
-MODIFIED_BEFORE date -MDB date	Same as -BEFORE option (added for uniformity of user interface)
-SAVED_AFTER date -SVA date	Reserved for future use.
-SAVED_BEFORE date -SVB date	Reserved for future use.
-RBF	Reserved for future use.

MAGLIB

MAGLIB, the library containing tape routines, which was part of MAGNET, is now a shared library. The following lines, therefore, must be added to the C_PRMO file in CMDNC0:

```
OPR 1
SHARE SYSTEM>ML2222 2222
RESUME SYSTEM>ML4000 1/16
OPR 0
```

The new C_PRMO.TEMPLATE is shown in the update to the System Administrator's Guide for Rev. 19.2, UPD5037-192.

LD

The output returned by the LD command has been heavily modified, and six new options have been added. For details, see the section on LD later in this chapter.

The file type field has been expanded to include some new file types, which are reserved for future use. Another field has been reserved for future use, and will be marked as "not set".

LOGIN

At Rev. 19.2, a process can take an abort while logging in. A new inactivity timeout has also been added, limiting the time a process can take to log in. For more details, see the section on LOGIN later in this chapter.

New LOTLIM Configuration Directive

LOTLIM specifies how long processes can take to log in. The format of this directive is:

LOTLIM n

where n is the number of minutes that PRIMOS will allow for a process to log in. The minimum value of n is 2, and the default is 3. It is recommended that most systems retain the default, which should provide enough time for users to type in a password and project id, and for PRIMOS to validate them.

LOGOUT: The command processor now prohibits a return to command level during a logout grace period.

USAGE: The parameters %ASYNC and %ICS now indicate CPU percentage required to support both ICS1 and ICS2 controllers.

STATUS: The STATUS COMM command now recognizes ICS2 controllers and displays the device address and number of asynchronous lines attached to each ICS2 on the system. For example:

OK, STATUS COMM

Controller	Type	Device Address	Lines Async	Sync
ICS2		11	32	0
AMLC	DMQ	54	16	0

New ICS INPQSZ Configuration Directive

ICS INPQSZ, a new cold start directive, has been introduced at Rev 19.2 to provide functionality similar to the AMLIBL directive for AMLCs. For details on the ICS INPQSZ directive, see Chapter 5 of this book.

New Configuration Error Message

A new error message now appears if NSLUSR is found to be too large at cold start.

- NSLUSR IS TOO BIG (BINIT)
  NSLUSR DEFAULTS TO ITS MAXIMUM VALUE: 63

The number of slave users specified by an NSLUSR directive exceeds the maximum number of configurable slave users ('77, decimal 63). Cold start continues, with the system configured for 63 slave users.

DOCUMENTATION CORRECTIONS

NLBUF Configuration Directive: Page 2-1 of the Revision 19.1 Software Release Document (MRU4304-009) misspells the directive as "NLBUFS". The correct version can be found on pages 1-31 and 3-13 of the System Administrator's Guide Update, Rev. 19.1 (UPD5037-191).

PAGDEV: If specified, the minimum value for records is '10 (8 decimal). See page 3-15 of the System Administrator's Guide Update, Rev. 19.1 (UPD5037-191).

Physical Device Numbers: Corrected versions of the table of "Physical Device Numbers for Storage Modules and Fixed Media Devices" can be found on pages A-4 and A-5 of the System Administrator's Guide Update, Rev.19.1 (UPD5037-191) and on pages D-6 and D-7 of the System Operator's Guide Update, Rev. 19.0 (UPD5038-190).

The following corrections apply to the System Administrator's Guide (DOC5037-190).

Lword: On page 8-12, the description of the lword option of the AMLC command should include the following additional note (POLER #47364):

Note

Bit 2 of lword is meaningful only if bit 1 is set.

Disks--Backup Considerations: In the first full sentence on page 6-6, change "...partitions of two, three, or five surfaces..." to "...partitions of one to five surfaces..."

The following corrections apply to the PRIMOS Commands Reference Guide (FDR3108-190).

DELAY Command: On page 2-42, under the entry for DELAY, insert at the end of the first paragraph (POLER #51311):

All parameters of the DELAY command must be specified in octal.

-BEFORE/-AFTER Date Formats: In response to a question about why the -BEFORE/-AFTER date formats are not more fully documented in the guide: Page 4-7 points the user to Chapter 18 of the Prime User's Guide (DOC4130-190). Explanation of date formats requires over two full pages; it is provided in one book and too long to be duplicated elsewhere (POLER #51252).

SOFTWARE PROBLEMS FIXED

USAGE: USAGE now handles 32-character user ids. User ids of up to eight characters are printed on the same line as the statistics; longer ones cause the statistics to be printed on the following line. Output with the -BRIEF option has not changed; only the first six characters of the user id are printed.

STATUS USERS: The user type of batch jobs now appears as "batch" instead of "phant".

Semaphores: Quit and restart while waiting on a numbered or named semaphore rewaits on the semaphore.

File Truncation: A problem occurred when a file was truncated on a modulo 1024 word boundary, leaving an empty record on the end of the file. In DAM files, this rendered the DAM index inaccurate. If the file was then extended again, further difficulties occurred. This problem has been corrected (POLER #53066).

DBG: The message "DBG.SAVE NOT FOUND IN CMDNC0" is now displayed if DBG.SAVE is not in CMDNC0 (POLER #51022).

Directory Passwords: A problem with the PASSWD command allowed the creation of a password containing blanks. The ATTACH command then prevented the use of the password to attach to the UFD. This problem has been corrected (POLER #43014).

CPL: A syntax error in an &ARGS statement raised an ACCESS_VIOLATION$ signal at 32(3)/7765 (referencing 0(3)/0). This problem has been corrected (POLER #40149).

BOOT_CREATE

NEW FEATURES AND CHANGES

BOOT_CREATE is a new CPL utility that calls MAGSAV to create magnetic tapes suitable for booting PRIMOS.

BOOT_CREATE opens and reads a list file containing the names of the files to be saved on the boot tape. The format required for the list file is described below.

BOOT_CREATE makes two passes of the list file. The first pass checks that the files exist and can be accessed. If there is a problem with either check, the option accompanying the pathname is evaluated and the appropriate message given. On the second pass, MAGSAV is called, and the files are written to tape.

The format of the command is:

▶ BOOT_CREATE [pathname] [options]

where pathname specifies the name of the file that lists the files and directories to be saved on the boot tape. This file can be stored anywhere on the system. If a filename is not given on the command line, the user is prompted for it later. A sample list file is included on the master disk for the guidance of users.

Options are as follows:

-HELP -H — This prints a help text giving the calling sequence and available options.

-NOPASS -NP — This suppresses any prompts for passwords. If any files or directories are protected by a password, this is logged as an error, and the program aborts at the end of the first pass.

-MT [drive] — This suppresses the drive number prompt. If -MT is specified without a number, drive 0 is assumed.

For further details on BOOT_CREATE, see Volume II of the System Operator's Guide (DOC7324-192).

COPY

SOFTWARE PROBLEMS OUTSTANDING

A user may get several questions on the same object, and see multiple and extra error messages, such as "ALREADY EXISTS", after answering no to a prompt from COPY. For example:

```
OK, COPY DAIMLER.KXM602 DAIMLER -DELETE
"DAIMLER" already exists, do you wish to overwrite it? n
Already exists.  "DAIMLER" (copy)
ER!
```

To merge directories and to move files into and out of a segment directory, use FUTIL rather than COPY.

COPY_DISK

NEW FEATURES AND CHANGES

COPY_DISK now sets the reserved words in the MFD and the ACL pointer to zero.

DOCUMENTATION CORRECTION

The following correction applies to the System Operator's Guide (DOC5038-190).

Handling Badspots: Page 7-11 states, "If the BADSPT file does not exist, specify MAKE to the badspots on both the FROM and the TO disks." This information is incorrect. For the correct procedure, see page 7-11 of the System Operator's Guide Update, Rev. 19.0 (UPD5038-190).

SOFTWARE PROBLEMS FIXED

Copying to Virgin Disk: COPY_DISK now copies to a virgin disk whether -NO_BADS is specified or not.

Copying from Virgin Source Partition: COPY_DISK does not copy from a virgin source partition; if it encounters one, it aborts with the message:

```
MFD HEADER IS CORRUPT OR NOT MADE ON PARTITION nn
bad format YOU CANNOT COPY FROM A VIRGIN DISK (BADADD)
```

CPL

NEW FEATURES AND CHANGES

Changes to the WILD Function

These arguments have been added to the WILD function:

-MODIFIED_AFTER date -MDA date	Same as -AFTER option (added for uniformity of user interface)
-MODIFIED_BEFORE date -MDB date	Same as -BEFORE option (added for uniformity of user interface)
-SAVED_AFTER date -SVA date	Reserved for future use.
-SAVED_BEFORE date -SVB date	Reserved for future use.
-RBF	Reserved for future use.

Change to the ATTRIB Function

This argument has been added to the ATTRIB function:

-DTS	Reserved for future use.

SOFTWARE PROBLEM FIXED

&ARGS Statement: A syntax error in an &ARGS statement raises an ACCESS_VIOLATION$ signal at 32(3)/7765 (referencing 0(3)/0) (POLER #40149).

Editor

SOFTWARE PROBLEM FIXED

MODE COUNT: The MODE COUNT command has been fixed. Previously, an abbreviated form of this command (such as MODE COU) would be treated as if it were a MODE NCOUNT command (POLER #43550).

EDIT_PROFILE

NEW FEATURES AND CHANGES

Initialization

System-wide and project-based groups are now always enabled on ACL systems. EDIT_PROFILE no longer asks you to specify what kinds of groups will use your system.

Limits

The efficiently manageable limit on the number of users who may belong to a project has changed from 2,000 to 20,000.

Prior to Rev. 19.2, the number of Project Administrators per system was limited to 24. At Rev. 19.2 there is no limit on the number of Project Administrators per system.

The efficiently manageable limit on the number of users per System Administration Directory (SAD) has changed from 5,000 to 21,000.

Conversion Procedure for EDIT_PROFILE

The data structures used by EDIT_PROFILE prior to Rev. 19.2 differ from those used at Rev. 19.2. In general, these data structures are upward (not downward) compatible.

Some early versions of EDIT_PROFILE did not set the version numbers correctly. SADs built with these early EDIT_PROFILES must be rebuilt before they can be read by 19.2 or later versions of EDIT_PROFILE.

Note

Rebuild any SAD created with a version of EDIT_PROFILE earlier than the 19.2.

The SADs should be rebuilt project by project using the EDIT_PROFILE subcommand:

```
REBUILD -PROJECT project-id
```

This should be done with EDIT_PROFILE Rev. 19.1. It must be done before users log in under Rev. 19.2.

For further details on EDIT_PROFILE, see the System Administrator's Guide Update, Rev. 19.2 (UPD5037-192).

SOFTWARE PROBLEM FIXED

LIST_PROJECT: The -OUTPUT option to the LIST_PROJECT subcommand now works correctly.

EMACS

NEW FEATURES AND CHANGES

The speed and performance of EMACS at Rev. 19.2 has been substantially improved. EMACS now comes up much faster. Also, because its working set is smaller, performance is improved in a multi-user environment.

Note

> Previous EMACS documentation suggested that you store commands that you write in EMACS*>LIB. If you are doing so, you should remove them and place them in one of your own directories. In the future, Prime may be overwriting and/or removing this subdirectory.

Enhancements to EMACS

At Rev. 19.2, the command libraries contained in EMACS* have been incorporated within EMACS. Consequently, you no longer need to use the load command. Similarly, there is no need go through a load-like procedure in library files when you are creating your own environments. (These procedures were discussed in Appendix A of the EMACS Reference Guide (IDR5026).

For compatibility, the load command remains within EMACS. However, it now has no effect.

The following library commands replace previously used fundamental commands:

Old Command	New Command
tab	type_tab
one_window	mod_one_window
split_window	mod_split_window
split_window_stay	mod_split_window_stay
write_file	mod_write_file

All these commands are discussed in the EMACS Reference Guide.

The older commands are still available if you wish to continue using them. All you need do is rebind them to a keypath in a library file.

For example:

```
(set_permanent_key "^X1" "one_window")
```

This example rebinds the one_window function to CONTROL-X1.

In addition, the elimination of the load command has made obsolete some of the information presented in Appendix A of the EMACS Reference Guide. For more details on the new changes, see EMACS, Rev. 19.2. (PTU2600-105).

RPG Mode

EMACS RPG mode now supports Prime's VRPG compiler as well as the RPG compiler. RPG and VRPG modes are identical for editing purposes. The difference is in the "compile" command.

To enable RPG mode for the VRPG compiler, use the "vrpg_on" command. To disable RPG mode, use the "rpg_off" command. When "vrpg_on" is in effect, the "compile" command invokes the VRPG compiler.

When "rpg_on" or "vrpg_on" is in effect, the current buffer is in overlay mode as well as in RPG mode.

Standard User Interface (SUI)

The HELP command within SUI is now called "sui_help" instead of "help". This does not affect users of the HELP function key.

The more advanced standard library tab facility has replaced the SUI tab facility. This menas that users of the original SUI tab facility using the "set_tabs" command are now prompted for a default tab interval. Five is no longer assumed.

DOCUMENTATION CORRECTION

The following correction applies to TEACH-EMACS, Prime's three-part on-line EMACS tutorial, available in the directory EMACS*.

TEACH-EMACS_1: Line 174 was inadvertently truncated. Change "{ESC}3" to "{ESC}3{ESC}B". This is the correct command to move backward three words (POLER #40225).

FIX_DISK

NEW FEATURES AND CHANGES

New Command Option

The command line option -LIST_BADSPOTS (-LB) causes FIX_DISK to list badspots and remapped records on the terminal. At Rev. 19.2, FIX_DISK no longer lists these items by default.

LABEL

NEW FEATURES AND CHANGES

LABEL has been modified to support both ANSI standard labels and PRIME non-standard labels.

LD

NEW FEATURES AND CHANGES

Changes in Output Format

The default output format lists selected entry names four or fewer to a line. If no entries are selected for output, the line "No entries selected" is output. Before printing entry names, LD will print a top header line, which contains:

- The pathname of the directory listed.
- The current user's access rights (in parentheses).
- The number of records used by this directory, if available.
- The number of records used by this directory and its subdirectories.
- The maximum quota set, if this is a quota directory.

Entries are grouped according to type, such as files, directories, or segment directories. An entry type header appears before each group of entries. Each of these headers shows how many entries of the particular type will be listed. If any attributes besides names are being displayed, the type header will also contain a label showing the output display format.

LD pauses output after every 23 lines, displays a --More-- prompt line, and awaits user response. By entering a Q, Quit, N, or No in upper or lower case characters, the user can suppress further output and leave LD. Any other user response directs LD to re-output the type header if output contains any attributes and to continue listing entries.

New Command Options

The -SORT_SIZE Option

The -SORT_SIZE option, abbreviated -SORTSZ, specifies that the entries be sorted by size in descending order within their types. If -SORT_SIZE is specified, -SORT_NAME, -SORT_DTM, or -SORT_DTS cannot also be specified. The sizes of entries other than access categories, and quotas of directory entries, are also displayed.

The -SORT_DTS Option

The -SORT_DTS option, abbreviated -SORTS, is reserved for future use.

The -BRIEF Option

The -BRIEF option, abbreviated -BR, specifies that the top header be output in a short, one-line format and that the column labels not be output. There will be no padding of the screen with blank lines before the paging prompt.

The -WIDE Option

The -WIDE option specifies that the output be displayed with a line width of more than 79 spaces. This option is particularly useful for line printer output.

For output of names only, LD will assume a line width of 100 spaces; for output with attributes, it will assume that all attributes besides the name will fit on one output line.

The -NO_WAIT Option

The -NO_WAIT option, abbreviated -NW, specifies that the display not pause after every 23 lines of output.

The -DTS Option

The -DTS option is reserved for future use.

Changes to Existing Command Options

Change to -NO_HEADER Option

The -NO_HEADER option, abbreviated -NHE, now suppresses the printing of both the top header and the entry type headers.

Change to -NO_SORT Option

The -NO_SORT option, abbreviated -NSORT, which specifies that the entries listed not be sorted, either by name or by type, now suppresses the printing of entry type headers.

Change to -SORT_DTM Option

The -SORT_DTM option, abbreviated -SORTD or -SORTM, which specifies that the entries be sorted by descending date/time modified within their types, is incompatible with the new -SORT_DTS or the -SORT_SIZE options.

Change to the -SORT_NAME Option

The SORT_NAME option, abbreviated -SORTN, which specifies that the entry names be sorted alphabetically, disregarding their types, is also incompatible with the new SORT_DTS or the -SORT_SIZE options. Type headers are not output.

Change to the -DETAIL Option

The -DETAIL option, abbreviated -DET, which specifies that all available attribute values be displayed for each entry selected, now displays the attributes of each entry on two lines following the entry name, unless the -WIDE option is also used. Attributes are displayed as follows:

On the first line:

1. Access rights available to this user (for password directories, the protection keys are displayed);

2. Size of entry in physical disk records (applies to files, directories, and segment directories only);

3. Delete-protection switch ("pr" if protected, "no" otherwise; applies to files, directories, and segment directories only);

4. Date/time modified;

5. Type of protection (name of access category protecting entry, either "(Specific)" for specific protected, or "(Default)" for protected by default; applies to files, directories, and segment directories only).

On the second line:

6. Type of entry (with logical type RBF indicated by a "-RBF" appended to entry's physical type);

7. Quota of entry in physical disk records (for directories only);

8. Setting of concurrency lock on entry ("sys" for system, "excl" for N readers or 1 writer, "updt" for N readers and 1 writer, and "none" for N readers and N writers; applies to files and segment directories only);

9. Incremental dump switch ("dmp" if the entry has been dumped, "no" otherwise; applies to files, directories, and segment directories only);

10. Date/time last saved (currently unused and displays "** not set **").

SOFTWARE PROBLEMS FIXED

Wrap Around: If LD output contained information about file system objects with long names, it was wrapped around. This no longer happens because of the new output format.

-DETAIL and -CATEGORY_PROTECTED: When these two options were both specified, random characters were displayed in the output. This problem has been corrected.

LOGPRT

NEW FEATURES AND CHANGES

New Event Types

NPXRLS: The NPXRLS event type was added at Rev. 19.1 to record bad virtual circuit clearing in NPX.

SENSOR: A new event type, SENSOR, has been added at Rev. 19.2. A SENSOR entry is written to the log file when a check from the environmental sensors on newer processors shuts down the system.

Change for New Central Processor

Machine check event handling has been modified to print a detailed interpretation of the DSW registers for newer processors.

New Commands

At Rev. 19.2, PRINT_SYSLOG and PRINT_NETLOG are new commands, which print system or network event logs, respectively . These commands are improved versions of LOGPRT and display clearer messages.

For further details on PRINT_SYSLOG and PRINT_NETLOG, see Volume II of the System Operator's Guide (DOC7324-192).

MAGLIB

NEW FEATURES AND CHANGES

MAGLIB as a Shared Library

MAGLIB, the library containing tape routines, which was part of MAGNET, is now shared. The following lines must be added to the file C_PRMO in CMDNC0:

```
OPR 1
SHARE    SYSTEM>ML2222 2222
RESUME   SYSTEM>ML4000 1/16
OPR 0
```

Error Recovery

MAGLIB has been modified to perform error detection and correction on all read or write operations.

SOFTWARE PROBLEMS FIXED

MT$WRIT and MT$READ: MT$WRIT was modified to append the length of the record to each record being written to tape. MT$READ was modified to return the length of the record when variable length records are being processed (POLER #46379).

MAGNET

NEW FEATURES AND CHANGES

The * extent variables have been redeclared with actual values. This was done to make MAGNET compatible with the new MAGLIB, described earlier in this chapter.

DOCUMENTATION CORRECTIONS

The following corrections apply to the Magnetic Tape User's Guide (DOC5027-183).

$UPDT: On page 8-4, the initial "$" symbol of the $UPDT command was inadvertently omitted.

Parity: In response to a question about the statement "Prime tape drives always write odd parity for nine-track tapes": The information on pages 1-10 and 1-11 is correct. Perhaps extra emphasis should be added to the distinction between nine-track mag tape parity and the so-called parity bit of ASCII.

The nine-track tape frame stores one eight-bit byte, plus one parity bit. The byte may be any data value from 0 to 255, possibly representing an ASCII character. The parity bit is odd, but is normally hidden from software, which sees only eight-bit bytes of data.

ASCII is a seven-bit character code stored in eight-bit bytes. The eighth bit is called the parity bit, from its former use in paper tape systems. Most computer manufacturers set this "parity" bit to space (always 0) or mark (always 1). Prime chooses mark. The value of this "parity" bit is unchanged by the medium upon which it is represented, whether mag tape, disk, or anything else. Some Prime software, however, forces this bit to mark. When an ASCII character is written to magnetic tape, the ASCII "parity" bit is part of the eight bits of data (POLER #58966).

MAGSAV/MAGRST

NEW FEATURES AND CHANGES

MAGSAV and MAGRST now set severity when run under PRIMOS. The severity is recorded and, if it is greater than zero, the ER! prompt is given. If, for example, MAGSAV or MAGRST is called from within a CPL file, the severity will be set after an error is encountered, and the program will then exit via the CPL severity handler.

MAGSAV and MAGRST do not set severity under PRIMOS II.

Pathnames

MAGSAV now recognises pathnames on commands $A and $I, which include disk names and passwords. The user can therefore attach away and use index files with either a full or qualified pathname reference.

The following formats can be used for both attaching and indexing:

```
<diskname>directory-name>subdirectory-name>etc
<diskname>directory-name password>subdirectory-name password>etc
          directory-name>subdirectory-name>etc
          directory-name password>subdirectory-name password>etc
```

BOOT_CREATE is a utility, implemented in CPL, that enables tapes suitable for booting from to be made up via a call or calls to MAGSAV. For further details, refer to the section on BOOT_CREATE earlier in this chapter.

SOFTWARE PROBLEMS FIXED

BADSPOTS: If a BADSPT file already exists, MAGRST will not attempt to restore it (POLER #40864).

Severity Handling: Severity handling is provided at Rev. 19.2, as described above (POLERS #45811, #45813).

Messages: All messages and warnings now appear in lowercase.

ACL and Password Directories: MAGRST now issues a warning message if a user tries to restore an ACL-protected directory while attached to a password-protected directory.

PRIMOS II: Under PRIMOS II, MAGSAV and MAGRST now assume the magnetic tape controller number to be zero.

MAKE

NEW FEATURES AND CHANGES

Badspot Handling

MAKE's ability to detect disk badspots has been greatly improved.

Command Interface

The command interface has changed substantially; it is now in the form of options instead of a dialogue.

Command Line Options

The -DISK Option

This option takes the form:

```
-DISK physical_device_number
```

This argument must be specified on the command line. The physical device number is that of the partition to be made. This argument replaces the question "PHYSICAL DEVICE:" at earlier revs.

The -PARTITION Option

This option takes the form:

```
-PARTITION partition_name
```

This argument is required; if it is not on the command line, MAKE prompts the user for it. A file with the same name as the partition is created on the MFD. This file is the record availability table. This argument replaces the question "PACK NAME?" in Rev. 19.1 and earlier.

The -DISK_TYPE Option

This option takes the form:

```
-DISK_TYPE disk_type
```

Use this option to specify a disk type other than the default. The default is disk type SMD. Other types are CMD, 68MB, 158MB, 160MB, 600MB, and FLOPPY.

The -MAP_UNCORR Option

This option takes the form:

-MAP_UNCORR

This option maps out only those records containing uncorrectable errors. Records containing correctable errors are treated as usable. Use of this option is not recommended.

The -SPLIT Option

This option takes the form:

-SPLIT number_of_paging_records

This option should be used to make a split partition. If the number of paging records is not typed on the command line, the total number of records available will be printed, and the user will be prompted for the number of paging records. This option replaces the question "SPLIT DISK?" in Rev. 19.1 and earlier.

The -FORMAT Option

This option takes the form:

-FORMAT

This option is used to write "hard" formats on a disk that has never been used on a Prime system.

The -QUERY_BADSPOTS Option

This option takes the form:

-QUERY_BADSPOTS

This option causes MAKE to query the user for known badspots on the disk.

The -PRE_REV19 Option

This option takes the form:

-PRE_REV19

This option should be used to make a partition for a system earlier than Rev. 19.

The -BADSPOT_LEVEL Option

This option takes the form:

-BADSPOT_LEVEL bad_spot_checking_level

This option specifies the amount of checking for badspots. Checking levels can be from 0 to 4 inclusive. If level 0 is specified, no checking is done. Level 4 gives the best checking, but takes the longest. The default is level 1 for SMD or CMD disk types and level 4 for fixed media disks (all others).

The -BAUD_RATE Option

This option takes the form:

-BAUD_RATE valid_baud_rate

This option is used to set the initial baud rate of the supervisor terminal. The default is 300. Other valid baud rates are 110, 1200, and 9600.

The -NO_INIT Option

This option takes the form:

-NO_INIT

When this option is used, the file system portion of the disk is not initialized. This option should be used only rarely.

For further details on MAKE, see Volume II of the System Operator's Guide (DOC7324-192).

PHYSAV/PHYRST

NEW FEATURES AND CHANGES

PHYSAV/RST has been modified to copy to a virgin disk without the specification "-NO_BADS". PHYSAV will not copy from a virgin disk and will abort with the message:

```
THE MFD HEADER IS CORRUPT OR NOT MADE ON <partition>
bad format YOU CANNOT COPY FROM A VIRGIN DISK badadd
```

PHYSAV will then abort.

SOFTWARE PROBLEMS FIXED

PHYRST--Restoring to a Different Partition: At Rev. 19.1, when PHYRST tried to restore a partition to another place, it tried to restore it to a partition that the user had not specified. This problem has been corrected.

PHYSAV--Saving Several Partitions: When saving several partitions that had not been input in head offset order, the cylinder limit was not set for any disks rearranged. As a result, PHYSAV saved only the first cylinder for those disks. All other disks were saved correctly.

PRINT_SYSLOG/PRINT_NETLOG

NEW FEATURES AND CHANGES

PRINT_SYSLOG and PRINT_NETLOG are new commands, which print system or network event logs, respectively. These commands are improved versions of LOGPRT and display clearer messages.

Neither PRINT_SYSLOG nor PRINT_NETLOG works under PRIMOS II. To print event logs when running under PRIMOS II, use LOGPRT.

Both commands require Rev. 19.2 or later PRIMOS.

For details on how to use these commands and the options available to them, see Volume II of the System Operator's Guide (DOC7324-192).

RUNOFF

DOCUMENTATION CORRECTIONS

The following corrections apply to the New User's Guide to EDITOR and RUNOFF (FDR3104-101).

.UNDENT/.RUNDENT: On page 6-4, add to the conclusion of the section on indentation (POLER #58984):

> The commands .U [m] and .RU [m] may not have a value for m that exceeds the margin limit.

Table of Contents: Page 7-11 states that a heading in the table of contents too long to fit on one line "will be split at a space and continued, indented, on the following line." The second line, however, is not indented. The example on the same page is also incorrect (POLER #44671).

Tabbing: Insert at the bottom of page 10-13 (POLER #53414):

> Whenever the TAB command appears at the beginning of a line of text, the rest of that line is neither filled nor adjusted. Filling or adjusting resumes with the next line.

SEG

NEW FEATURES AND CHANGES

SHARE Command Enhanced

The SHARE command now not only allows indentation of the file identification for CPL files, but this file prefix can now be from one to twenty-eight characters in length, making share file names less cryptic.

Performance Improvements

The speeds of the LOAD sequence and some maps have been improved.

New Warning and Commands

A new warning is given for a subsequent use of a common block that is smaller than a previous one. This is to warn the user of a possible mismatch in data between these data structures. When a smaller definition of a common block occurs, the message:

ILLEGAL REDEFINITION OF COMMON

is given with the common block name.

If you want to suppress this warning, the command NSCW (No Smaller Common Warning) suppresses further checking and SCW (Smaller Common Warning) reenables the test.

DOCUMENTATION CORRECTION

The following correction applies to the LOAD and SEG Reference Guide (PDR3524-172).

MIX and SPLIT: Page 4-7 states, "In general, loading under the MI option will reduce the number of segments required for a program, but debugging such programs may be more difficult." Program size will not be reduced, however, when MIX is used along with SPLIT (POLER #33176).

SOFTWARE PROBLEMS FIXED

Base Areas: The problem of overwriting base areas generated by the AUTOMATIC command by successive small procedures has been fixed.

NEW: The NEW command in the MODIFY subprocessor did not work if the file being read in was read-only protected, and at some other times. Both these problems were corrected (POLERS #48535,48536).

SHARE: The problem that the response to SHARE command could not be indented has been corrected by the enhancement to SHARE described above (POLER #34326).

SOFTWARE PROBLEM OUTSTANDING

Naming Problem: There is no warning of common and procedure having the same name (POLER #43595).

SLIST

NEW FEATURES AND CHANGES

Performance Improvement

The execution of SLIST has been speeded up.

SORT

SOFTWARE PROBLEM FIXED

Pathnames: SORT now handles full pathnames correctly.

SPOOL

SOFTWARE PROBLEM FIXED

Phantom Request Time-out Handling: All PROP requests made to the spooler phantom now wait 40 rather than 30 seconds before being timed-out if the phantom fails to respond.

Subroutines

NEW SUBROUTINES

Add the following new subroutines to Chapter 10:

▶ TTY$IN

Purpose

This function checks whether there are any characters in the user's TTY input buffer. The state of the buffer is undisturbed by the call: no character is actually read or removed from the buffer.

Usage

```
DCL TTY$IN ENTRY () RETURNS (BIT(1)ALIGNED)

more-to-read = TTY$IN
```

more-to-read Will be true ('1'b) if there is at least one character of input available at the terminal of the calling process, and '0'b otherwise.

Discussion

TTY$IN is used to check whether there is at least one character of input currently available on the calling process' terminal. Use TTY$IN when you do not want to wait for input via a call to CL$GET, ClIN$, or TlIN. TTY$IN allows the program to poll for input and perform other processing while waiting for input to arrive.

If TTY$IN is called in a non-interactive process, '0'b is always returned, whether or not a command input file is active.

It is possible for TTY$IN to return '1'b, and for a subsequent call to ClIN$ to wait for input. This can happen if the user types Control-P after TTY$IN is called, which causes a quit to PRIMOS and the flushing of the input buffer. When the user types START, the next call to ClIN$ will then wait for a character.

TTY$IN is necessary at Rev. 19 to cut down on CPU usage. Before Rev. 19, checks of the input buffer could be done only with an R-mode routine that, at Rev. 19, has a high overhead of CPU usage. Use of TTY$IN can cut CPU usage by half.

Because FTN cannot call subroutines with no argument, this routine may not be called directly from FTN. To get the benefits of the routine, use an F77 or PMA interlude.

Command Error Reporting

This discussion applies to the two subroutines which follow, SETRC$ and SS$ERR.

When a command or subsystem detects an error situation, two parties must in general be notified: the user, who is usually interactive, and the invoker, which is simply the procedure that invoked the command or subsystem. Typically, the user is notified by means of a diagnostic message, whereas the invoker must be notified by a method more suitable for programmed decisions—a status code.

The requirement that subsystems be able to keep control on errors if interactive but give up control if noninteractive is met by requiring subsystems to call the routine SS$ERR. Use of SS$ERR is necessary to support the Command Procedure Language product. Without it, CPL is not able to support its documented error handling features fully because it does not receive proper indication of compilation, loading, and file handling failures.

Severity Codes

A severity code is a single FIXED BINARY(15,0) value in which two distinct pieces of information may be encoded. First, the severity level has the value 0, -1, or +1; this is the arithmetic sign of the severity code. Second, the absolute value of the severity code may (or may not) be a standard error code, as defined below. The meaning, if any, of the absolute value of a severity code must be interpreted relative to the specific command that returned it: the same absolute value returned by two different commands may not mean the same thing.

The meanings of the severity level of the severity code, however, are the same no matter which command returned the code. They are as follows:

0 No errors—execution successful.

-1 Warning(s)—minor exceptions encountered, but the results of the command's execution are usable to the best of the command's ability to determine.

1 Error(s)—serious errors encountered. Some of the results of the command are not usable, or some of the actions requested could not be performed.

When a command or command function has decided to return control to its caller, it must also return a severity code value if it encountered an error. Command callers initialize the severity code to 0 before calling a command so that the command need take no action if no errors are encountered.

If the procedure is part of a user-created program, it should use the primitive SETRC$ to return the severity code.

Standard Error Codes

A standard error code is always to be interpreted according to some error table. Error tables are identified by 32-character names. At present, only the PRIMOS error table exists, accessible via ERRPR$. It is assigned the null name. See Appendix E, "Error Handling for I/O Subroutines."

A standard error code is a compact representation of a diagnostic message and is usually returned by a command or subroutine to its caller. This code identifies the precise cause of an error encountered by the callee. A standard error code is converted to a severity code by changing its arithmetic sign to the proper severity level value.

Subsystem Error Handling

Whenever a conversational subsystem encounters an error in the syntax of a subcommand or during its execution and that subsystem wishes to returns to its own command level, it must:

1. Print any applicable diagnostics;
2. Call the PRIMOS subroutine SS$ERR (subsystem error);
3. Return to its command level;
4. Not return a positive severity code when it finally returns control to PRIMOS, since then the user would see an ER! prompt when he is not expecting one.

When a subsystem encounters an error and immediately returns to PRIMOS without going back to its own command level, it does not make any difference whether the subsystem is being used interatively or not. Hence the subsystem should:

1. Print any applicable diagnostics;
2. Call SETRC$ to set a positive (or negative) severity code as appropriate;
3. Return to PRIMOS. The user will see an ER! prompt, if interactive, or a CPL procedure will receive the proper error code, if not.
4. SS$ERR should not be called in this case.

SS$ERR works approximately as follows. When called, SS$ERR checks whether the user is "interactive", that is, whether the process is a non-phantom whose command input stream is connected to the terminal. If so, SS$ERR simply returns. Otherwise, SS$ERR raises the condition SUBSYS_ERR$. The default handling of this condition is for the command processor to abort the subsystem via a nonlocal goto back into the command processor, where a positive severity code is forced.

Users and subsystem implementors should keep the following points in mind:

- The user's program may make an on-unit for SUBSYS_ERR$ which simply returns. This causes SS$ERR to return to the subsystem as if the user were interactive, thus defeating the noninteractive abort mechanism. (This option would rarely be useful.)

- The subsystem may use the condition mechanism's CLEANUP$ condition to regain control in one last gasp before the nonlocal goto is completed. (For details on the condition mechanism, see Chapter 22.) This will allow the subsystem to perform any required cleanup activities before it actually loses control.

Subsystems should call SS$ERR after printing diagnostics and before returning to their command level if they intend to retain control.

Subsystems should not call SS$ERR if they will return to PRIMOS immediately on the error.

Calling Sequences

▶ SETRC$

```
dcl SETRC$ entry(fixed bin, bit(1)aligned);

call SETRC$(severity_code, abort_flag);
```

severity_code is the severity code to return to the invoker of this program. (Input)

abort_flag is '1'b if the command file (if any) is to be aborted, and '0'b if it is not to be aborted. (This flag will make no difference if this command was invoked by a CPL procedure.) (Input)

If the severity_code is less than or equal to 0, then abort_flag is ignored, and the command file is never aborted. If abort_flag is omitted from the calling sequence, it is assumed to be '0'b.

SETRC$ was described in the Revision 19.1 Software Release Document (MRU4304-009). The description is repeated here for the convenience of the reader.

▶ SS$ERR

```
dcl SS$ERR entry ();

call SS$ERR;
```

There are no arguments.

If the caller is being used interactively, SS$ERR simply returns. Otherwise, the condition SUBSYS_ERR$ is raised, which usually results in the termination of the caller by means of a nonlocal goto back to the command processor.

▶ ERTXT$

This routine accepts a standard PRIMOS error code and returns the character string representation of its error message as it would be printed by the routine ERRPR$. Its declaration and calling sequence follow:

```
dcl ERTXT$ entry(fixed bin, char(1024)var);

call ERTXT$(code, errmsg);
```

code Standard error code. (Input)
errmsg Text of error message. (Output)

Add the following to page A-28:

▶ DIR$SE

Purpose

This new routine replaces and extends the functionality of DIR$LS.

DIR$SE is a general purpose directory searcher that returns entries meeting caller-specified selection criteria.

Usage

```
dcl dir$se entry (fixed bin, fixed bin, bit(1), ptr, ptr,
                  fixed bin, fixed bin, fixed bin,
                  (4) fixed bin, fixed bin, fixed bin);

call dir$se (dir_unit, dir_type, initialize, sel_ptr,
             return_ptr, max_entries, entry_size,
             ent_returned, type_counts, max_type, code);
```

dir_unit	unit on which directory to be searched is open (Input)
dir_type	type of object open on dir_unit (Input)
initialize	If set, directory is to be reset to the beginning. If unset, it is to be searched from the current position. (Input)
sel_ptr	pointer to structure containing selection criteria (see below) (Input)
return_ptr	pointer to caller's return structure for selected entry data (see below) (Input)
max_entries	maximum number of entries to be returned (should be greater than zero unless this routine is being used only to initialize the directory) (Input)
entry_size	number of words to be returned per entry (Input)
ent_returned	number of entries returned (Output)
type_counts	number of entries of each type returned in the order: dirs, seg dirs, files, access categories (This argument should be a 4-word array.) The type_counts are incremented each time DIR$SE is called, i.e., the number of types returned in this call of DIR$SE is added to the current type-counts totals. When the "initialize" bit is set, these counts are reset to the total number of types returned in this call. (Input/output)
max_type	number of types in type_counts (currently must be 4) (Input)
code	standard error code (Output)

Possible values are:

e$bver	bad version number for selection criteria structure (currently can only be zero (0))
e$bpar	bad max_type (currently must be 4)
e$eof	There are no more entries in the directory to be selected.
e$stl9	Selection criteria involving date/time last saved or RBF file type have been specified, and the PRIMOS rev that accesses the directory does not support these features.

The selection criteria should be supplied in the following structure. The "sel_ptr" parameter should point to this structure.

```
dcl 1 selection_criteria based,
      2 version_no fixed bin,
      2 wild_ptr ptr,
      2 wild_count fixed bin (15),
      2 desired_types,
        3 dirs bit(1),
        3 seg_dirs bit(1),
        3 files bit(1),
        3 access_cats bit(1),
        3 RBF bit(1),
        3 spare bit(11),
      2 modified_before_date_time fixed bin (31),
      2 modified_after_date_time fixed bin (31),
      2 saved_before_date_time fixed bin (31),
      2 saved_after_date_time fixed bin (31),
```

where

version_no — Must be zero for this version of the selection criteria structure.

wild_ptr — If wildcard entryname selection is to be applied to the directory entries, this field should point to a list of wildcard names for which to search. The list should be an array of char(32) varying strings, and the names must be in upper case. Wildcards are explained in the Prime User's Guide (DOC4130-190).

wild_count — Is the number of names in the list pointed to by wild_ptr. If wild_count is zero, entryname is not used as a selection criterion.

desired_types — A bit-encoded field defining which types of directory entries the caller wishes to have returned. The first four bits of this field specify the physical types of the entries that are to be returned. The fifth bit can be used in combination with the other four bits to select entries that are also RBF entries, and thus have a logical type of '1'. To select only RBF segment directories, the seg_dirs and RBF bits should both be set, and the other bits not set. If the first four bits are set, all entries will be returned. If all five bits are set, all entries that are also RBF entries will be returned.

modified_before_date	Selects entries with date/time modified earlier than this date. The date should be in standard FS format (described with routine CV$DQS). Should be zero if this field is not to be used as a selection criterion.
modified_after_date	Selects entries with date/time modified later than this date. The date should be in standard FS format (described with routine CV$DQS). Should be zero if this field is not to be used as a selection criterion.
saved_before_date	Reserved for future use. Must be zero currently.
saved_after_date	Reserved for future use. Must be zero currently.

DIR$SE will return the information for all the entries selected by this call in the following structure:

```
dcl 1 dir_entries (*) based,
      2 ecw,
        3 type bit (8),
        3 length bit (8),
      2 entryname char(32) var,
      2 protection,
        3 owner rights,
          4 spare bit (5),
          4 delete bit (1),
          4 write bit (1)
          4 read bit (1),
        3 delete_protect bit (1),
        3 non_owner_rights,
          4 spare bit (4),
          4 delete bit (1),
          4 write bit (1),
          4 read bit (1),
      2 file_info,
        3 long_rat_hdr bit(1),
        3 dumped bit(1),
        3 dos_mod bit (1),
        3 special bit (1),
        3 rwlock bit (1),
        3 spare bit (2),
        3 type bit (8),
      2 date_time_mod fixed bin (31),
      2 non_default_acl bit (1) aligned,
      2 logical_type fixed binary,
      2 trunc bit (1) aligned,
      2 date_time_last_saved fixed bin (31);
```

where

ecw	Entry control word for the entry:	
	type: 2	normal directory entry (file, directory or segment directory)
	3	access category
	length:	24 words for PRIMOS revs up to and including 19.2; 27 words for PRIMOS revs from 19.2 onwards.
entryname	name of the entry	
protection	owner_rights	Are the rights granted to a user when attached to the containing directory with owner rights.
	delete_protect	If this bit is set, the file may not be deleted. The bit may be reset by a call to the SATR$$ routine.
	non_owner_rights	Are the rights granted to a user when attached to the containing directory with non-owner rights.
file_info	long_rat_hdr	If set, indicates that the file is a Disk Record Availability (DSKRAT) file spanning more than one disk record.
	dumped	If set, this file has been saved by MAGSAV and has not been modified since then.
	dos_mod	If set, this file was modified while PRIMOS II (DOS) was running. It indicates that the date/time last modified field may be incorrect.
	special	If set, this is a special file (e.g., DSKRAT, BOOT, MFD) and may not be deleted.

rwlock — Indicates the setting of the file's read/write concurrency lock.

Possible values are:

0 system default setting
1 unlimited readers or one writer (exclusive)
2 unlimited readers and one writer (update)
3 unlimited readers and writers (none)

type — Indicates the type of object described by this entry.
Possible values are:

0 SAM file
1 DAM file
2 SAM segment directory
3 DAM segment directory
4 Ufd
6 Access category

date_time_mod — The date/time the file was last modified, in standard FS format. FS format dates are described with routine CV$DQS.

non_default_acl — This bit is set if the object is not protected by the default ACL—that is, if it is protected by a specific ACL or by an access category.

logical_type — This is an additional file type to the physical file type described in file_info.type.
Possible values are:

0 for normal files
1 for RBF files

trunc — This bit is set if the file has been truncated by the FIX_DISK utility; otherwise, reset to zero.

date_time_last_saved — Reserved for future use. This field will currently be returned as zero (unset).

Example

The following program will list all entries in a UFD whose names end in the suffix ".PLP" and which were modified after December 1, 1982 and before January 1, 1983.

```
main:
        proc;

$INSERT syscom>keys.ins.pll

dcl 1 bvs based,
      2 len fixed bin,
      2 chars char(32)

/* Arguments to/from srch$$ */

dcl dir_unit fixed bin,          /* Unit on which UFD is open */
    dir_type fixed bin,          /* Type of entry open on unit
                                    dir_unit */
    code fixed bin;              /* Error return code */

/* Data structures for dir$se */

dcl initialize bit (1),         /* Initialize/no initialize UFD
                                   switch */
    selec_ptr ptr,              /* Pointer to selection criteria
                                   structure */
    return_ptr ptr,             /* Pointer to output structure
                                   containing selected entries */
    max_entries fixed bin,      /* Max number of entries that can
                                   be selected per call to DIR$SE */
    entry_size fixed bin,       /* Maximum directory entry length */
    ent_returned fixed bin,     /* Number of entries returned for
                                   this call to DIR$SE */
    type_counts (4) fixed bin,  /* Total of types returned so far by
                                   DIR$SE */
    max_type fixed bin,         /* Size of type_counts array */
    wild_cards (1) char(32)var, /* Entryname sel. criteria */

    1 dir_entries (10),         /* Array of 10 directory entries */
       2 etype bit(8),          /* Dir entry type */
       2 elength bit(8),        /* Entry length in words */
       2 entryname char(32)varying,
       2 prot,                  /* Access control info */
          3 owner bit(8),
          3 non_owner bit(8),
       2 file_info,
          3 (long_rat_hdr, dumped, dos_mod, special) bit(1),
          3 rwlock bit(2),
          3 padl bit(2),
          3 type bit(8),
       2 dtm fixed bin(31),
       2 non_default_acl bit (1) aligned,
```

```
      2 log_type fixed bin,
      2 trunc bit (1) aligned,
      2 dts fixed bin(31),
    dir_entries_ptr ptr,

    1 selec_crit,                  /* Structure containing selection
                                      criteria */
      2 version_no fixed bin,      /* Version number for input and
                                      output data structure */
      2 wild_ptr ptr,              /* Pointer to wildcard names
                                      structure */
      2 wild_count fixed bin (15), /* Number of wild names */
      2 types,                     /* File types to be selected */
        3 dirs bit (1),             /* Physical file types */
        3 seg_dirs bit (1),
        3 files bit (1),
        3 acl bit (1),
        3 rbf bit (1),              /* Logical file type */
        3 spare bit (11),
      2 dtmb fixed bin (31),       /* Date/time modified selection -
                                      before date */
      2 dtma fixed bin (31),       /* - after date */
      2 dtsb fixed bin (31),       /* Must be zero */
      2 dtsa fixed bin (31);       /* Must be zero */

/* Other local variables */

dcl first bit (1),                 /* True if first call to DIR$SE
                                      in loop */
    i fixed bin,
    date char(128) var;

/* External entry points */

dcl DIR$SE entry (fixed bin, fixed bin, bit(1), ptr, ptr, fixed bin,
        fixed bin, fixed bin, (4) fixed bin, fixed bin, fixed bin),
    IOA$ entry options(variable),
    CV$DTB entry (char(128) var, fixed bin(31), fixed bin),
    ERRPR$ entry (fixed bin, fixed bin, char(*), fixed bin, char(*),
        fixed bin),
    SRCH$$ entry options(variable);

/* Open current directory on unit 10 */

        dir_unit = 10;
        call SRCH$$ (k$read, k$curr, 0, dir_unit, dir_type, code);
        if code ^= 0
           then call ERRPR$(k$irtn, code, '', 0, '', 0);

/* Directory open OK, set up selection criteria for entries
   to be selected */

        else do;
```

```
/* Select entries modified between 1st December 1982
   and 1st January 1983 */

          date = "01/01/83.00:00:00";
          call CV$DTB(date, selec_crit.dtmb, code);
          if code ^= 0
             then call TNOU("Modified before date error.", 26);

          date = "12/01/82.00:00:00";
          call CV$DTB(date, selec_crit.dtma, code);
          if code ^= 0
             then call TNOU("Modified after date error.", 25);

/* Select entries whose names end in the .PLP suffix */

          selec_crit.wild_ptr = addr (wild_cards);
          selec_crit.wild_count = 1;
          wild_cards(1) = "      .PLP";

/* Select only file entries */

          selec_crit.types.dirs = '0'b;
          selec_crit.types.seg_dirs = '0'b;
          selec_crit.types.acl = '0'b;
          selec_crit.types.files = '1'b;
          selec_crit.types.rbf = '0'b;

/* Set up other constants required for selection criteria structure */

          selec_crit.version_no = 0;
          selec_crit.dtsb = 0;                    /* Must be zero */
          selec_crit.dtsa = 0;                    /* Must be zero */
          selec_ptr = addr (selec_crit);

/* Set up parameters to DIR$SE for return data */

          dir_entries_ptr = addr (dir_entries);
          max_entries = 10;     /* Handle 10 entries at a time */
          entry_size = 27;
          max_type = 4;

/* Call DIR$SE to return 10 entries at a time which satisfy the
   selection criteria */

          first = '1'b;      /* Initialize/no initialize switch */

          call TNOUA('Entries selected:  .', 21); /* Output hdr */

          do while (code = 0);
             if first
                then initialize = '1'b;          /* Initialize UFD
                                        if first call to DIR$SE */
             else initialize = '0'b;
```

```
        call DIR$SE (dir_unit, dir_type, initialize, selec_ptr,
                     dir_entries_ptr, max_entries, entry_size,
                     ent_returned, type_counts, max_type, code);

         first = '0'b;        /* Do not initialize UFD on next
                                 call to DIR$SE */

         do i = 1 to ent_returned;          /* Output names of
                                            entries selected */

            call TNOU(addr(dir_entries (i).entryname)->bvs.chars
                      length(dir_entries (i).entryname))
            end;
         end;
      end;
   end;
```

DOCUMENTATION CORRECTIONS AND ADDITIONS

The following corrections apply to the Subroutines Reference Guide (DOC3621-190).

Quad Precision

In the table of data types on page 5-2, add a new row for 128-bit quad precision. This type is implemented in FORTRAN 77 as REAL*16 and in PMA in the format nnQnn.

On page 5-7, add the following paragraph:

> REAL*16
>
> This is a quad precision floating-point number, implemented as a 128-bit value. It corresponds to the PMA format nnQnn. It is described in detail in the Rev. 19.2 edition of the FORTRAN 77 Reference Guide (DOC4029-192).

In the table of data types on page 8-2, add a new row for 128-bit quad precision. This type is implemented in FORTRAN 77 as REAL*16 and in PMA in the format nnQnn.

On page 8-5, add the following paragraph:

QUAD PRECISION

This is a quad precision floating-point number, implemented as a 128-bit value. It corresponds to the PMA format nnQnn, but can be passed to and from FORTRAN 77 only as a REAL*16 number. For details, see the Rev. 19.2 Assembly Language Programmer's Guide (PTU2600-104).

STRING Data Type

In the table of data types on page 6-3, add the Pascal data type STRING to the "Varying character string" horizontal row.

On page 6-6, add the following paragraph to the discussion of the CHARACTER(*)VARYING type:

The STRING data type in Prime Pascal is equivalent to the PL1G CHARACTER(*)VARYING type. A Prime extension to Pascal, the STRING type is new at Rev. 19.2. For information on passing Pascal strings to PL1G CHARACTER(*)VARYING strings and vice versa, see Appendix D, "Interfacing Pascal to Other Languages", in the Pascal Reference Guide Update, Rev. 19.2 (UPD4303-192).

K$POSR: On page 9-20, change the definition of K$POSR to:

Moves the file pointer of funit by the number of words specified by pos relative to the position achieved after rwkey is performed.

RDEN$$: In the description of buffer on page 9-28, delete the second sentence, concerning a key of 3.

File Entry Format: In Figure 9-1 on page 9-30, change "18 RESERVED" to "18 NON-DEFAULT ACL."

In the description of the Entry Control Word on the same page, delete the values :000001 and :000424, and add the following discussion:

:001430 Type=3, length=24. A type of 3 indicates an access category UFD entry. All the above information is returned.

NON_DEFAULT_ACL: On page 9-31, after the PROTEC listing, add the following:

NON_DEFAULT_ACL: The high-order bit is 1 if this UFD entry is protected by a specific ACL or access category, 0 if it is protected by the default ACL. Bits 2-16 are reserved.

In the values under FILTYP on the same page, add:

6 Access category.

Read/Write Lock: In the example on page 9-33, delete the line:

```
IF (TYPE.NE.1.AND.TYPE.NE.2) GOTO 100 /* UNKNOWN
```

On page 9-34, delete line 10: "rden_buffer(23)..."

On page 9-35, delete line 15: "rden_buffer(23)..."

SATR$$: Add the following new keys to the SATR$$ routine on page 9-39:

K$LTYP Set the logical type field in the file entry to the value in data_array(1). This field should not be set by user software. It is for Prime internal use only.

K$DTLS Reserved for future use.

K$TRUN Set the "truncated by FIX_DISK" bit from the value in bit 1 of data_array(1). This field should not be set by user software. It is for Prime internal use only.

SRSFX$: In the description of SRSFX$ on page 9-56, the argument suffix-list should be described as (*)CHAR(32)VAR, as it can be a structure of variable strings.

A pathname of '' (null string) will open the current UFD.

Here is an example of a simple program that uses SRSFX$ to check on the existence of a file. It also uses the CL$PIX routine.

```
main:
       proc;

$Insert syscom>keys.ins.pl1

$Insert syscom>errd.ins.pl1

/* External entry points */

dcl srsfx$ entry (fixed bin, char(*)var, fixed bin, fixed bin,
        fixed bin, (1) char(32)var, char(32)var, fixed bin,
        fixed bin),
    cl$get entry (char(*)var, fixed bin, fixed bin),
    cl$pix entry (bit(16) aligned, char(*)var, ptr, fixed bin,
        char(*)var, ptr, fixed bin, fixed bin, fixed bin, ptr),
    errpr$ entry (fixed bin, fixed bin, char(*), fixed bin, char(*),
```

```
        fixed bin),
    tnoua entry (char(*), fixed bin),
    todec entry (fixed bin),
    tnou entry (char(*), fixed bin);

/* Local declarations */

dcl 1 bvs based,                        /* Based Varying String */
      2 len fixed bin,
      2 chars char (128);

Dcl pathname char(80)var,
    dir_name char(80)var,
    fil_name char(80)var,
    unit fixed bin,
    type fixed bin,
    num_suff fixed bin,
    suff_list (10) char(32)var,
    suff_used fixed bin,
    status fixed bin,
    code fixed bin,
    non_st_code fixed bin,
    pix_index fixed bin,
    bad_index fixed bin,
    picture char(30)var,
    pic_ptr ptr,
    out_ptr ptr,
    arg_line char(150) var;

dcl 1 args,
      2 dir char(128) var,
      2 file char(32) var;

/* PROMPT USER FOR ARGUMENTS */

call tnoua('Enter directory pathname and filename arguments:', 49);

/* READ IN ARGS TO CALL */

call cl$get (arg_line, 150, code);
if code ^= 0
   then call errpr$(k$nrtn, code, 'CANNOT READ ARGS', 16, 'test', 9);

   else do;

/* SET UP DATA FOR CL$PIX */

      picture = 'tree; entry; end';
      pic_ptr = addr(picture);
      out_ptr = addr(args);

/* CALL CL$PIX TO PARSE ARGUMENTS */

      call cl$pix(0, 'test', pic_ptr, 30, arg_line, out_ptr,
```

```
            pix_index, bad_index, non_st_code, null());
      if non_st_code ^= 0
         then do;
         call tnoua ('CANNOT PARSE ARGS, error code = ', 32);
         call todec (non_st_code);
         call tnou(' ', 1);
         end;

      else do;

/* CHECK FOR EXISTENCE OF FILE IN SON, FATHER, GRANDFATHER ORDER */

         unit = 2;
         num_suff = 3;
         suff_list(1) = '.SON';
         suff_list(2) = '.FATHER';
         suff_list(3) = '.GRANDFATHER';

         pathname = dir || '>' || file;
         call srsfx$(k$exst, pathname, unit, type, num_suff,
                  suff_list, file, suff_used, status);

         if status > 0
            then call errpr$(k$irtn, status, addr (pathname) ->
                     bvs.chars, length (pathname), '', 0);

         else do;
            if suff_used = 0
               then do;
               call tnoua('base file name only found: ', 27);
               call tnou(addr(pathname) -> bvs.chars,
                           addr(pathname) -> bvs.len);
               end;
            else do;
               pathname = pathname || suff_list(suff_used);
               call tnoua (addr(pathname) -> bvs.chars,
                           addr(pathname) -> bvs.len);
               call tnou (' form of file name found', 24);
               end;

            end;
         end;
      end;
   end;
```

This program will give the following output if the '.SON' form of the file exists.

```
R TEST
Enter directory pathname and filename arguments: TEST_UFD TEST_FILE
TEST_UFD>TEST_FILE.SON form of file name found
```

CL$PIX: On pages 10-6 and 10-8, change the argument "code" to "non-std-code".

On page 10-19, add the following example for CL$PIX. It is a simple program that uses CL$PIX to parse a command line.

```
test:
       proc;

/* EXTERNAL ENTRY POINTS */

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

/* INSERT FILES */

$Insert syscom>keys.ins.pll

/* LOCAL DECLARATIONS */

dcl code fixed bin,                  /* standard error code */
    non_st_code fixed bin,           /* cl$pix error code */
    pix_index fixed bin,
    bad_index fixed bin,
    picture char(30) var,
    pic_ptr ptr,
    out_ptr ptr,
    arg_line char(150) var;

dcl 1 args,
      2 dir char(128) var,
      2 file char(32) var;

dcl 1 bvs based,
      2 len fixed bin,
      2 chars char(1);

/* PROMPT USER FOR ARGUMENTS */

call tnoua('Enter directory pathname and filename: ', 38);

/* READ IN ARGS TO CALL */

call cl$get (arg_line, 150, code);
if code ^= 0
   then call errpr$(k$nrtn, code, 'CANNOT READ ARGS', 16,
        'test', 9);
```

```
else do;

/* SET UP DATA FOR CL$PIX */

   picture = 'tree; entry; end';
   pic_ptr = addr(picture);
   out_ptr = addr(args);

/* CALL CL$PIX TO PARSE ARGUMENTS */

   call cl$pix('3'b3, 'test', pic_ptr, 30, arg_line, out_ptr,
            pix_index, bad_index, non_st_code, null());
   if non_st_code ^= 0
      then do;
      call tnoua('CANNOT PARSE ARGS, error code = ', 32);
      call todec(non_st_code);
      call tnou(' ', 1);
      end;

/* OUTPUT ARGUMENTS READ IN */

   else do;
      call tnoua('Directory pathname = ', 21);
      call tnou(addr(dir) -> bvs.chars, addr(dir) -> bvs.len);

      call tnoua('File name = ', 12);
    call tnou(addr(file) -> bvs.chars, addr(file) -> bvs.len);
      end;
   end;
end;
```

The above program gives the following output.

```
Enter directory pathname and filename:
<testpk>my_ufd my_file
Directory pathname = <TESTPK>MY_UFD
File name = MY_FILE
```

PHNTM$: On page 10-35, add to the description of the argument filename:

> The filename must end in ".CPL" if the program is a CPL program. Non-CPL programs must not have a .CPL suffix.

CMDL$A: In the sample program on page 12-16, the line number 2 should be 20.

FEDT$A: In the description of FEDT$A on page 12-27, the date field should have the format 'DAY, DD MON YEAR'. The function returned is 'DD.MM.YY'.

NLEN$A: In the description of NLEN$A on page 12-37, the argument namlen should be defined as "length of the variable name."

RNAM$A: On page 12-46, the data type of the argument name is ASCII. This argument must begin with a nonnumeric character that is also not a plus or minus sign.

RTRN$S: Lines 2 and 3 of the definition of length on page 13-23 should read:

When all records have been returned, calls to RTN$$ return a record length of 0.

CLNU$S: After "& TYP" on page 13-24, insert a new variable:

```
& NIL    /*Dummy variable for ignored return values
```

In line 1 on page 13-25, replace the argument (0,0) with (NIL,NIL).

BNSRCH: On page 13-31, when the key opflag is 2 and the new item is not found, INDEX is set to 0 before the return.

Logical Devices, Physical Devices, and File Units: On page 14-8, the last two lines of Table 14-3 should be:

140	MPC printer 0
141	MPC printer 1

I$AP02: For a discussion of the arguments for I$AP02, which are not mentioned on page 18-6, see Chapter 14.

TIDEC, TIHEX, TIOCT, TNOU, TNOUA, TODEC, TOHEX, TOOCT, TONL: The arguments variable and count on pages 18-10 through 18-13 are all INTEGER*2.

SPOOL$: In words 4-6 of the information array on page 19-9, if no forms option is specified, the space should be filled with blanks.

T$AMLC: The data type of the argument stat-vec on page 20-21 should be defined as FIXED BIN(31).

SLEEP$: The argument interval on page 21-24 must be expressed as a multiple of 100 milliseconds.

ARITH$ Condition: Add to the explanatory paragraph on page 22-26:

This condition is raised by fixed overflow or zero divide.

FINISH Condition: Change the last sentence of the explanatory paragraph on page 22-30 to:

In PL1G, a STOP statement causes FINISH to be raised after files are closed. In this case, FINISH also raises the STOP$ condition.

STORAGE Condition: Change the last sentence of the explanatory paragraph on page 22-40 to:

In PL1G, the STORAGE condition can be raised either through the ALLOCATE statement or by the compiler making its own call.

STRING-3 Condition: On page 22-40, add the STRING-3 condition, raised if that compile option is used in PL1G.

SUBSYS_ERR$ Condition: On the same page, add the following error condition:

SUBSYS_ERR$

The subroutine SS$ERR raises this condition when it is called by a subsystem that is not interactive (i.e., one run by a CPL or command file). The default on-unit for SUBSYS_ERR$ aborts execution of the subsystem and forces the severity code to have a positive sign. Any command input file is aborted.

DIR$LS: Add to the definition of the argument type counts on page A-24:

The type-counts fields are incremental, showing the number of entries for each type so far. At Rev. 19.2, they are reset to zero when the initialize bit is set.

Add the following to the discussion of DIR$LS on page A-25:

The directory entry structure returned to the user has been extended at PRIMOS Rev. 19.2. The complete structure is as follows:

```
dcl 1 dir_entry,
      2 ecw,
        3 type bit (8),
        3 length bit (8),
      2 entryname char(32) var,
      2 protection,
        3 owner rights,
          4 spare bit (5),
          4 delete bit (1),
          4 write bit (1)
          4 read bit (1),
        3 delete_protect bit (1),
        3 non_owner_rights,
          4 spare bit (4),
          4 delete bit (1),
          4 write bit (1),
          4 read bit (1),
      2 file_info,
        3 long_rat_hdr bit(1),
        3 dumped bit(1),
        3 dos_mod bit (1),
        3 special bit (1),
        3 rwlock bit (1),
        3 spare bit (2),
        3 type bit (8),
      2 date_time_mod fixed bin (31),
      2 non_default_acl bit (1) aligned,
      2 logical_type fixed binary,
      2 trunc bit (1) aligned,
      2 date_time_last_saved fixed bin (31);
```

The length field is 24 words up to and including Rev. 19.1 and 27 words from Rev. 19.2 onwards. The new fields added are:

logical_type	This is an additional file type to the physical file type described in file_info.type. Possible values are: 0 for normal files 1 for RBF files
trunc	This bit is set if the file has been truncated by the FIX_DISK utility; otherwise, it is zero.
date_time_last_saved	Reserved for future use. This field is now returned as zero (unset).

DIR$RD: Add the following to the discussion of DIR$RD on page A-27:

The directory entry structure returned to the user has been extended. The complete structure is as follows:

```
dcl 1 dir_entry based,
      2 ecw,
        3 type bit(8),
        3 length bit(8),
      2 name char(32),
      2 pw_protection bit(16) aligned,
      2 non_default_protection bit(1) aligned,
      2 file_info,
        3 long_rat_hdr bit(1),
        3 dumped_bit bit(1),
        3 dos_mod bit(1),
        3 special bit(1),
        3 rwlock bit(2),
        3 spare bit(2),
        3 type bit(8),
      2 date_time_modified,
        3 date,
          4 year bit(7),
          4 month bit(4),
          4 day bit(5),
        3 time fixed bin,
      2 logical_type fixed bin,
      2 reserved fixed bin,
      2 trunc bit(1) aligned,
      2 date_time_last_saved fixed bin (31);
```

The new fields added are:

logical_type	This is an additional file type to the physical file type described in file_info.type. Possible values are: 0 for normal files 1 for RBF files
trunc	This bit is set if the file has been truncated by the FIX_DISK utility; otherwise, it is zero.
date_time_last_saved	Reserved for future use. This field is now returned as zero (unset).

ENT$RD: Add the following to the discussion of ENT$RD on page A-29:

The directory entry structure returned by ENT$RD has been extended. It is the same as for the routine DIR$RD.

CHAPTER 3

LANGUAGES

BASIC/VM

DOCUMENTATION CORRECTIONS

The following correction applies to The BASIC/VM Programmer's Guide (FDR3058-101).

Maximum Array Size: On page 9-1, insert at the end of the first paragraph (POLER #58299):

An array in BASIC/VM can have a maximum of 32766 elements.

The following correction applies to the BASIC/VM Programmer's Guide Update, Rev. 19 (COR3058-002). This update package should also have included page 14-1A, which was part of the Rev. 18.0 update package. Its contents, which should be inserted into the Rev. 19.0 update package, are:

▶ CHANGE str-expr TO num-array

Transforms ASCII character string, str-expr, into a one-dimensional numeric array (num-array) containing the decimal values of the ASCII codes of the string numeric array of ASCII codes to its string equivalent, str-var. ASCII characters and their decimal equivalents are listed in Appendix B. For example, if the string "WORD", (A$), is changed to array A,

This is page 14-1A in its entirety. The description of CHANGE continues on page 14-2.

SOFTWARE PROBLEMS FIXED

Rounding: A problem with rounding errors has been corrected (POLER #44455).

MIDAS: A MIDAS locking problem has been corrected (POLER #59082).

A problem with the MIDAS ON ERROR and ON SIZE conditions has been corrected (POLER #58987).

Procedure Calls: Invoking calls to odd length operating system procedures sometimes resulted in linkage faults. This problem has been corrected (POLER #57839).

COBOL

SOFTWARE PROBLEMS FIXED

LOW-VALUES: The compiler failed to flag as an error the move of the figurative constant "LOW-VALUES" to a binary (usage COMP) data name. Furthermore, the code generated would move an incorrect value (octal 177520) to the target item. The compiler has been changed so that when "LOW-VALUES" is moved to a binary (usage COMP) item, the value 0 (in binary) is moved to the target field. The compiler does not flag this move as an error.

The documentation of permissible moves should be changed to allow LOW-VALUES to be moved to a binary item (POLER #27319).

READ Statement: A READ statement of a RELATIVE file in a program containing large data records (usually where the link frame size was greater than 32767) raised an ACCESS_VIOLATION$ signal. This problem has been corrected (POLER #58982).

SOFTWARE PROBLEMS OUTSTANDING

Link Frame: The compiler shows incorrect link frame sizes in program statistics for some very large programs (POLER #20775).

UNSTRING Statement: If the receiving item of an UNSTRING statement has a picture clause greater than 1023, such as PIC X(1024), the compiler terminates with an INTERNAL ERROR 106 or 112 (POLER #32692).

LEVEL 88 Conditions: When subscripted LEVEL 88 conditions that have been defined in a COMP table are used, the compiler generates incorrect code, and a reference to the LEVEL 88 name does not evaluate correctly. For example:

```
01 GROUP.
   05 NAME PIC 9 COMP OCCURS 5 TIMES.
      88  COND-NAME VALUE 1.
.
.
.
   IF COND-NAME(I) ...
```

The problem can be avoided by taking the COMP out of the description at the 05 level (POLER #32693).

IF Statements: Certain combinations of AND/OR in IF statements do not work correctly (POLER #35031).

SOURCE LEVEL DEBUGGER

NEW FEATURES AND CHANGES

Support for Pascal STRING Type

The Source Level Debugger offers full support to Prime Pascal's new STRING data type at Rev. 19.2. The STRING data type is similar to the CHARACTER VARYING type in PL/1G. For more information on STRING, see the Pascal section of this document or the Pascal Reference Guide Update, Rev. 19.2 (UPD4303-192).

Support for FORTRAN 77 REAL*16 (Quad Precision)

The Source Level Debugger offers full support to FORTRAN 77's new REAL*16 (quad precision) data type at Rev. 19.2. For more information on REAL*16, see the FORTRAN 77 section of this document or the Third Edition of the FORTRAN 77 Reference Guide (DOC4029-192).

The AGAIN Command

A new debugger command, AGAIN, is available at Rev. 19.2. The AGAIN command, abbreviated A, causes the debugger command immediately preceding it to be repeated. The format of the AGAIN command is:

```
AGAIN
```

Here is an example using the AGAIN command with the LANGUAGE and : (evaluation) commands:

```
> LANGUAGE
Language is PASCAL.
> AGAIN
Language is PASCAL.
> : X
X = 7
> AGAIN
X = 7
```

FTN

SOFTWARE PROBLEMS FIXED

Floating-point Constants: At Rev. 18.4, the compiler was modified to convert floating-point constants (e.g., "-3.2766") more accurately. At Rev. 19.2, the I/O runtime library support module, F$IOFTN, has been modified to use a more accurate conversion algorithm that is consistent with the way FTN now converts constants (POLER #56395).

DBG: Setting breakpoints on statements following certain IF statements resulted in unexpected behavior within DBG when FTN programs were debugged. This problem has been corrected.

FORTRAN 77 (F77)

NEW FEATURES AND CHANGES

Quadruple Floating-point Arithmetic

F77 has been enhanced to allow quadruple floating-point arithmetic in FORTRAN programs on all 50 Series and up Prime computers. This enhancement is not supported on the Prime 400 at Rev. 19.2.

The compiler supports a full set of Quad intrinsic functions and the REAL*16 data type. However, quadruple complex variables (COMPLEX*32) and intrinsic functions are not supported at Rev. 19.2.

For further information, see the new edition of the FORTRAN 77 Reference Guide (DOC4029-192).

-PBECB Compile-time Option

F77 now supports the -PBECB option, which duplicates the functionality of the same option in the FTN compiler.

-PBECB causes F77 to place the Entry Control Block of every subprogram it compiles into the procedure frame, except for BLOCKDATA subprograms, which do not have an ECB. The compiler ignores this option when compiling a main program since it always puts a main program's ECB into the link frame.

This option is especially useful for large FORTRAN programs that are comprised of many subprograms and that will be shared on the system. Users running programs compiled with the -PBECB option generally have smaller working sets and demand less of the system paging resources.

Program Constants and Short (One Word) Instructions

The compiler has been enhanced to make determinations about the placement of program constants (such as PARAMETER constants) in the procedure code and their accessibility by short, rather than long, instructions.

Prior to Rev. 19.2, F77 placed all program constants at the beginning of the procedure frame. They had to be accessed by a long instruction if, for example, they were referenced at the end of a moderately sized FORTRAN program. F77 now places such constants within reach of their

point of reference by a short instruction if it finds that it can do so. This change should improve execution speed of certain FORTRAN programs and should also somewhat reduce their size.

DOCUMENTATION CORRECTIONS

The following corrections apply to the FORTRAN 77 Reference Guide (DOC4029-183).

Opening a File on a File Unit: On page 4-7, the second sentence under this heading should read:

> Every file except the user terminal, which is always open on FORTRAN unit number 1, must be connected to a file unit prior to data transfer.

OPEN Statement Options: On page 4-11, the first paragraph in Table 4-2, "OPEN Statement Options", should read:

> The file is opened on the FORTRAN unit number specified.

ENDFILE Statement: On page 4-17, in the first paragraph under ENDFILE, change "file unit unit#" to "FORTRAN unit unit#".

READ Statement: On page 4-19, in the first paragraph under READ, change "file unit 1, the terminal" to "FORTRAN unit 1, the terminal".

These corrections derive from POLER #40958.

SOFTWARE PROBLEMS FIXED

Floating-point Constants: These constants are now evaluated more precisely within F77. This improvement, made at Rev. 19.1, was erroneously reported on page 3-17 of the Revision 19.1 Software Release Document (MRU4304-009) as an "Outstanding Problem" (POLER #36539).

Double-precision Constants: The compiler can now handle double-precision constants in which the exponent has more than 2 digits, such as "1.0D200". This improvement, made at Rev. 19.1, was erroneously reported on page 3-17 of the Revision 19.1 Software Release Document as an "Outstanding Problem" (POLERs #29335, 32927).

TYPE: TYPE statements of the form <TYPE>*<LENGTH>, where LENGTH is not legal for the particular TYPE (for example, INTEGER*3 or REAL*11) now

generate a severity 3 error message for illegal length specifications (POLER #36579).

LOGICAL*4: The results of certain LOGICAL*4 expressions were not passed correctly to procedures. This occurred when the actual expression was passed instead of a variable that had been assigned the result of the expression, such as "CALL SUBR(I.EQ.J)". The compiler now generates proper code for this construct (POLERs #40617, 34287).

CHARACTER: Certain substrings of CHARACTER variables failed to be passed correctly to procedures, when passed as expressions rather than as variables, as in "CALL SUBR(CHAR(2:5))". The compiler has been fixed to generate proper code for this construct (POLERs #45973, 44294, 36012, 57176).

I/O Statements: The statements "READ(1.75) ..." or "WRITE(1.75) ..." were not flagged as errors by the compiler, but resulted in program failure when executed. F77 now issues a severity 3 error message for non-INTEGER unit specifications in I/O statements (POLERs #36707, 48000, 47598).

Listings: The compiler options listed at the top of the listing file produced by F77 failed to reflect whether the -FRN option was in effect or not. The options section has been re-organized and enhanced to reflect all pertinent options, whether on or off (POLER #48028).

Error Message 335: The token spelling of the offending token referenced in error message 335 is now printed out in that message (POLER #000100).

Long CHARACTER Variables: F77 produced an inappropriate error message, "BEGINNING OF FILE (ERROR_MESSAGE)...", when compiling certain programs. F77 did not deal effectively with a compiler limitation that disallows initialization of CHARACTER variables whose length is greater than 255 in a DATA statement. The compiler now produces an appropriate severity 2 error message when it encounters this situation (POLERs #40703, 40400, 44587).

WARNING 408: The first character of WARNING 408 was being overwritten by a single quote, ', in certain situations. This problem has been corrected (POLER #33000).

Range Checking: Range checking on array subscripts in I/O implied DO-loops did not work unless the compiler's optimization was also turned off, by use of the -NOOPT option, for example. Range checking now works for I/O implied DO-loops under all conditions (POLER #49403).

END: Program units containing only an END statement caused the F77 compiler to abort with an ACCESS_VIOLATION$ signal. This problem has been corrected (POLER #40000).

Subprogram Arguments: The actual length of certain subprogram arguments of type CHARACTER was not being properly determined at program runtime when the associated dummy arguments were declared as CHARACTER*(*). This was due to improper code emission by the compiler for this particular case. The problem has been corrected (POLER #35440).

PARAMETER Statements: F77 disallowed the definition of symbolic names in a PARAMETER statement that had previously been declared as CHARACTER*(*). Also affected was the usage of CHARACTER*(*) statement functions. This problem has been corrected (POLER #35199).

Two-Character Strings: The FORTRAN statement "STOP 99" printed only the second character of "99" when executed. The same held true for any two-character string, such as "AB". NAMELIST block names of two characters also exhibited the same problem, in that "AB" would actually have been known as "B" when compiled. Both of these problems have been corrected (POLERs #41473, 43214, 57405, 57406, 58924).

DBG: DBG incorrectly reported the types of certain REAL*4 arrays that had been passed to a subprogram. This problem has been corrected (POLER #56372).

Arrays: A LOGICAL*1 array was being initialized incorrectly by the compiler when initialization was specified by a DATA statement. This problem has been corrected (POLER #57407).

Implied DO-Loops: The compiler no longer aborts when parsing I/O implied DO-loops associated with variables of type CHARACTER. Statements like the following were involved:

```
WRITE(1,*) (CHAR(I:I),I=1,5)
```

SHFT Intrinsic: SHFT intrinsic now functions correctly when used with a negative second argument.

Error messages: Some error messages have been rewritten for greater clarity.

Single Precision Floating-point Constants: The compiler has been modified to convert single precision floating-point constants (e.g., "-3.2766") more accurately. To preserve consistency, the I/O runtime library support module, F$IO77, has also been modified to use a more accurate conversion algorithm.

SOFTWARE PROBLEMS OUTSTANDING

DATA Statements: Initialization of certain variables and arrays in DATA statements is performed incorrectly by the compiler. Affected are CHARACTER*1 arrays and odd length CHARACTER variables (POLERs #35864, 45126, 43025, 43203, 54458).

Function Subprograms: Error 444, "STATEMENT ORDERING CONFLICT", is issued at inappropriate times in certain function subprograms that have a length specification in their FUNCTION statements, e.g., "INTEGER*4 FUNCTION..."(POLER #41811).

-XREF Option: The use of -XREF when compiling certain programs in -DEBUG mode produces an ACCESS_VIOLATION$ signal in DBG, when attempting to evaluate variables in COMMON blocks (POLERs #43211, 46662).

List-directed I/O: List-directed I/O produces an incorrect number of blank lines (POLERs #37812, 54751).

Large COMMON Blocks: Various problems occur with certain large COMMON blocks, sometimes containing segment-spanning arrays (POLERs #44313, 48395, 59191).

Array Subscripts: Problems exist with certain mixed-mode array subscripts. One problem involves a CHARACTER array subscript (POLERs #40695, 47139, 32841, 34423).

An array subscript of zero is reportedly not allowed in a DATA statement (POLER #34423).

Octal Constants: Problems with octal constants occurred—one in DATA statement initialization, the other in a mixed-mode expression (POLERs #42727, 45632).

ENTRY Statements: Various problems occur with ENTRY statements. Two of these report DBG problems with setting breakpoints (POLERs #37173, 40892, 44321, 46975).

Equivalence: Error 218 is produced when a variable is equivalenced to itself (POLER #32259).

ENVIRONMENT

The F77 compiler for Rev. 19.2 must be run on a Rev. 19.2 PRIMOS operating system installed on a 50-Series and up computer, as it requires support for the new quadruple floating-point arithmetic feature. This support is provided by the UII package that is installed as part of 19.2 PRIMOS, and by Rev. 19.2 of the shared VFTNLIB and library files.

Rev. 19.2 of the compiler will operate on the Prime 400 only if FORTRAN users take great care not to use the new quadruple floating-point arithmetic feature. The Rev. 19.2 UII package, and quadruple floating-point arithmetic, will not operate on the Prime 400.

PASCAL

NEW FEATURES AND CHANGES

Several new features were added to the Pascal compiler at Rev. 19.2. The most significant enhancement is the new STRING data type. Pascal's STRING type allows easy manipulation of varying-length character strings, similar to PL/1-G's CHARACTER VARYING type. The STRING type is a Prime extension.

Two compiler options, -NOOPT1 and -NOOPT3, are new at Rev. 19.2.

New features of Pascal at Rev. 19.2 are also documented in the Pascal Reference Guide Update, Rev.19.2 (UPD4303-192). This update contains changes for the new features and some additional corrections.

The STRING Type

The STRING data type is a Prime extension. Similar to the PL1G CHARACTER VARYING type, the STRING type makes it easy to manipulate character strings in Prime Pascal. Unlike an array of characters, which must contain a precise number of character elements, STRING allows you to assign, compare, concatenate, read, write, and pass character strings that have a varying number of elements.

Declaring Strings

A variable of type STRING is declared in this form:

```
VAR
  string-identifier : STRING[n];
```

The string-identifier is the variable of STRING type, and n is the maximum number of character elements allowed in the string. This number is called the maximum length of the string. If n is not given in a STRING declaration, the maximum length is 80 by default.

Consider the following example:

```
VAR
  A : STRING;  {80 characters}
  B : STRING[5];  {5 characters}
  C : STRING[10];  {10 characters}
BEGIN
  B := 'HI';
  C := 'HELLO';
  WRITELN(C)
END.
```

The maximum length of string A is 80 characters. Strings B and C have maximum lengths of 5 and 10, respectively. During execution, at the WRITELN statement, B contains two characters and C contains five characters. Therefore, variables declared as type STRING can hold character-string values of any length less than or equal to the maximum length of the string. The length of a character string assigned to a STRING variable is called the operational length of the string. Thus, in the example above at the WRITELN statement, string B has a maximum length of 5 and an operational length of 2. String C has a maximum length of 10 and an operational length of 5. The operational lengths may change when new values are assigned to the character strings.

You can use CONST and TYPE declarations with STRING. For example:

```
CONST
  STRING_LENGTH = 20;
TYPE
  STRING_2 = STRING[2];
  STRING_5 = STRING[5];
  STRING_20 = STRING[STRING_LENGTH];
VAR
  ST2 : STRING_2;
  ST5 : STRING_5;
  ST20 : STRING_20;
```

Note

A string can be declared to have a maximum length of 32767 characters and a minimum length of 1 character.

The Null String

A null string, which is specified by '', is allowed. Null strings can be used to initialize strings. You may assign a null string, but an attempt to write a null string will generate a runtime error. The null string is also a Prime extension. Here is an example of a null string assignment:

```
VAR
  S : STRING[10];
BEGIN
  S := '';
```

Assigning Strings

Strings can be assigned to one another. When the value of one string is assigned to another string, the operational length is also assigned.

Note

A character literal string consists of one or more characters enclosed in single quotes. It should not be confused with a string, which is a variable that represents a STRING type value. Character literal strings, such as 'HELLO' or 'greetings', may be assigned to strings.

Here is an example that assigns character literals to strings and assigns one string to another string:

```
VAR
  ST2 : STRING[2];
  ST5 : STRING[5];
BEGIN
  ST2 := 'HI';
  ST5 := 'HELLO';
  ST5 := ST2      {operational length of ST5 is 2}
                  {and its value is 'HI'}
END.
```

If the operational length being assigned is larger than the maximum length of the string receiving the assignment, the excess characters are truncated. For example:

```
VAR
  ST2 : STRING[2];
  ST5 : STRING[5];
BEGIN
  ST5 := 'HELLO';
  ST2 := ST5      {value of ST2 is now 'HE'}
END.              {and its operational length is 2}
```

Here is another example of string assignments:

```
CONST
  STR_LENGTH = 10;
VAR
  A : STRING;
  B : STRING[4];
  C : STRING[8];
  D : STRING[STR_LENGTH];
BEGIN
  B := 'four';       {operational length is 4}
  B := 'fo';         {operational length is 2}
  D := '1234567890';      {operational length is 10}
  D := '12345';           {operational length is 5}
  D := B; {value of D is 'fo'}
  D := '123456';
  B := D; {value of B is '1234'}
  A := ''; {this is a legal assignment}
  WRITELN(A) {but this will cause a runtime error}
END.
```

Two rules govern string assignments:

- If the operational length of the string being assigned (the sending string) is less than or equal to the maximum length of the receiving string, then the entire string value is assigned, and the receiving string assumes the operational length of the sending string.

- If the operational length of the sending string is greater than the maximum length of the receiving string, then only the number of characters in the sending string equal to the maximum length of the receiving string are assigned. The remaining characters are not assigned.

Assigning Arrays and Strings to Each Other

Strings and arrays of characters can be assigned to one another through the use of two functions, STR and UNSTR. The STR function converts an array of characters or a single character to a string, and the UNSTR function converts a string to an array of characters or to a single character. The STR and UNSTR functions are Prime extensions.

The result of the STR function is a string with a length of the same number of characters as the array of characters argument. The result of an STR function may be used anywhere a string may be used.

The result of the UNSTR function is an array of characters or a single character. The number of characters in the newly formed array is determined by context. That is, the context of whatever array length is expected determines the length. The result of an UNSTR function may be used anywhere an array of characters is expected. Here are some specific rules governing the use of the UNSTR function:

- If the result of the UNSTR function is being assigned to an array of characters, then that result will have the same number of characters as the receiving array of characters.

- If the result of the UNSTR function is being passed to a procedure or function, then that result will have the same number of characters as the formal parameter.

- If the result of the UNSTR function is being compared to an array of characters, then that result will have the same number of characters as the array of characters to which it is being compared.

- If the UNSTR function is used in any other context, the length of the resultant array will be the same as the operational length of the string argument.

Here is an example that converts strings and arrays of characters to one another using STR and UNSTR:

```
VAR
  ST4 : STRING[4];
  ST8 : STRING[8];
  AR4 : ARRAY[1..4] OF CHAR;
  AR8 : ARRAY[1..8] OF CHAR;
BEGIN
  AR4 := 'JUNK';
  ST4 := STR(AR4);  {value of ST4 is 'JUNK'}
  AR4 := 'BLUE';
  ST8 := STR(AR4);  {value of ST8 is 'BLUE'}
  AR8 := 'LAVENDER';
  ST4 := STR(AR8);  {value of ST4 is 'LAVE'}
  ST4 := 'JUNK';
  AR4 := UNSTR(ST4);  {value of AR4 is 'JUNK'}
  AR8 := UNSTR(ST4);  {value of AR8 is 'JUNK'}
  ST8 := 'LAVENDER';
  AR4 := UNSTR(ST8)  {value of AR4 is 'LAVE'}
END.
```

Comparing Strings

String comparisons are allowed according to the following rules:

- If the strings have the same operational length, a normal comparison operation will be done.
- If the operational lengths of the strings are different, blanks will be assumed to follow the shorter string.

Here is an example that compares strings:

```
VAR
  ST4 : STRING[4];
  ST8 : STRING[8];
BEGIN
  ST4 := 'BLUE';
  ST8 := 'LAVENDER';
  IF ST8 > ST4 THEN
    WRITELN('Pass')  {this will pass}
  ELSE
    WRITELN('Fail');
  ST8 := ST4;        {ST8 is now 'BLUE'}
  IF ST8 = ST4 THEN
    WRITELN('Pass')  {this will pass}
  ELSE
    WRITELN('Fail')
END.
```

Concatenating Strings

Prime Pascal's concatenation operator (+) concatenates two strings into one string. The concatenation operator is a Prime extension. There is no concatenation operator in standard Pascal.

The resultant length of the newly formed string equals the sum of the operational lengths of the two concatenated strings. Either or both of the strings may be a character literal string.

Here is an example that uses concatenation:

```
VAR
  ST2 : STRING[2];
  ST4 : STRING[4];
  ST6 : STRING[6];
  ST11 : STRING[11];
  AR2 : ARRAY[1..2] OF CHAR;
  AR4 : ARRAY[1..4] OF CHAR;
  AR6 : ARRAY[1..6] OF CHAR;
BEGIN
  ST2 := 'HI';
  ST4 := 'BALL';
  AR2 := 'GO';
  AR4 := 'BLUE';
  ST6 := ST2 + ST4;  {ST6 equals 'HIBALL'}
  ST6 := ST4 + ST2;  {ST6 equals 'BALLHI'}
  ST4 := ST2 + ST4;  {ST4 equals 'HIBA'}
  ST11 := ST2 + ST4 + 'HELLO';  {ST11 equals 'HIHIBAHELLO'}
  ST4 := ST2; {ST4 equals 'HI'}
  ST4 := ST2 + ST4;  {ST4 equals 'HIHI'}
  ST6 := STR(AR2) + STR(AR4);  {ST6 equals 'GOBLUE'}
  AR6 := UNSTR(STR(AR4) + STR(AR2));  {AR6 equals 'BLUEGO'}
  ST2 := 'PA';
  ST4 := 'SCAL';
  ST6 := ST2 + ST4;
  IF ST6 = 'PASCAL' THEN
    WRITELN('Pass')  {this passes}
  ELSE
    WRITELN('Fail');
  IF ST6 = ST2 + ST4 THEN
    WRITELN ('Pass again')  {this passes}
  ELSE
    WRITELN ('Fail');
  IF ST6 = 'PA' + 'SCAL' THEN
    WRITELN ('This works too')  {this passes}
  ELSE
    WRITELN ('Fail');
  AR6 := UNSTR(ST6);
  IF AR6 = 'PASCAL' THEN
    WRITELN ('Passes to array')  {this passes}
  ELSE
    WRITELN ('Array fails');
  IF AR6 = UNSTR(ST2 + ST4) THEN
    WRITELN ('Passes to array again')  {this passes}
  ELSE
    WRITELN ('Array fails')
END.
```

The concatenation operator is also discussed in Chapter 7.

Reading and Writing Strings

When reading a string, you can enter any number of characters up to the maximum length. Consider the following program, which contains a READ statement:

```
VAR
  ST10 : STRING[10];
BEGIN
  READ(ST10)
END.
```

If the input were:

```
ABC(carriage return)
```

the program would assign 'ABC' to ST10 when the carriage return is entered.

If the input were:

```
ABCDEFGHIJK
```

the program would complete execution the moment the 'K' character was typed, because the 'J' character is the tenth character.

When you use a READLN statement, the number of characters before the carriage return becomes the operational length of the string up to the maximum length of that string.

Consider the following example:

```
VAR
  ST5 : STRING[5];
BEGIN
  READLN(ST5)
END.
```

If the input were:

```
ABC(carriage return)
```

the value of ST5 would be 'ABC' and ST5 would have an operational length of 3 characters.

If the input were:

```
ABCDE(carriage return)
```

or

```
ABCDEFGHIJKLM(carriage return)
```

the value of ST5 would be 'ABCDE' and the operational length of ST5 would be 5 characters. In either case, the program would not terminate until the carriage return was typed.

When reading two strings with one READ or READLN statement, you must enter all of the characters of the first string, up to its maximum length, before you can begin entering characters for the second string. Consider the following example:

```
VAR
  ST1, ST2 : STRING[10];
BEGIN
  READ(ST1, ST2)
END.
```

If the input were:

```
ABCDEFGHIJKLM
```

the characters 'ABCDEFGHIJ' would be assigned to ST1, and 'KLM' would be assigned to ST2. In order to assign characters to ST2, 10 characters must be assigned to ST1.

If you enter less than 10 characters, or if you enter only 10 characters, then the null string is assigned to ST2. (Null strings cannot be written out.)

When a string is written, the default field width is the operational length of the string. If a field width is specified, and the width of the field to be printed is greater than the operational length of the string, then the string is right justified in the field and blank padded on the left. If the specified field width is too small, then only the specified number of characters will be printed.

Here is an example of writing strings with different field widths:

```
VAR
  ST10 : STRING[10];
BEGIN
  ST10 := 'ABCDEFGH';  {eight characters}
  WRITELN(ST10);
  WRITELN(ST10:12);
  WRITELN(ST10:2)
END.
```

The output will look like this:

```
ABCDEFGH
    ABCDEFGH
AB
```

Here is another example that reads and writes strings to and from the terminal and PRIMOS data files:

```
VAR
  ST5 : STRING[5];
  ST10 : STRING[10];
  STRINGINPUT : FILE OF CHAR;
  STRINGOUTPUT : FILE OF CHAR;
BEGIN
  WRITE('Enter an ST5 value: ');
  READLN(ST5);
  WRITELN(ST5);
  WRITE('Enter an ST10 value: ');
  READLN(ST10);
  WRITELN(ST10);
  WRITELN(ST5 + ST10);
  RESET(STRINGINPUT, 'STINPUT');
  READLN(STRINGINPUT, ST5);
  REWRITE(STRINGOUTPUT, 'STOUTPUT');
  WRITELN(STRINGOUTPUT, ST5);
  READLN(STRINGINPUT, ST10);
  WRITELN(STRINGOUTPUT, ST10);
  WRITELN(STRINGOUTPUT, ST5 + ST10);
  CLOSE(STRINGINPUT);
  CLOSE(STRINGOUTPUT)
END.
```

Passing Strings to Procedures and Functions

Strings can be passed as parameters to procedures and functions. They may be passed by value or by reference and may return as arguments from functions.

The STRING assignment rules, given earlier in this chapter, apply to passing strings to procedures and functions.

Here is an example that passes strings to procedures and functions:

```
TYPE
  STRING_6 = STRING[6];
  STRING_3 = STRING[3];
  STRING_10 = STRING[10];
VAR
  GLOBAL_10 : STRING_10;
  GLOBAL_6 : STRING_6;
PROCEDURE PROC1(S : STRING_6);  {GLOBAL_10 is passed to S}
  BEGIN                         {and is truncated to 'TESTIN'}
    WRITELN(S)                  {'TESTIN' will be written}
  END;
PROCEDURE PROC2(VAR S : STRING_6);  {GLOBAL_10 is assigned to}
  BEGIN                             {the parameter GLOBAL_6}
    S := GLOBAL_10
  END;
FUNCTION FUNC(S : STRING_6) : STRING_3;  {GLOBAL_10 becomes}
  BEGIN                                  {substring 'TIN'}
    FUNC := SUBSTR(S, 4, 3)              {inside function}
  END;
BEGIN {main}
  GLOBAL_10 := 'TESTING';
  PROC1(GLOBAL_10);
  PROC2(GLOBAL_6);
  WRITELN(GLOBAL_6);  {'TESTIN' will be written}
  GLOBAL_10 := FUNC(GLOBAL_10);
  WRITELN(GLOBAL_10)  {'TIN' will be written}
END.
```

For complete information on procedures and functions, see Chapter 9 of the Pascal Reference Guide.

String Functions

There are seven other built-in functions that manipulate strings in addition to the STR and UNSTR functions. All of these functions are Prime extensions. They are:

- LENGTH
- INDEX
- SUBSTR
- DELETE
- INSERT
- TRIM
- LTRIM

The LENGTH Function: This function takes a string as an argument and returns an integer that is the operational length of the string. A string literal may not be used with this function.

The INDEX Function: This function takes two strings as arguments. It searches the first string to determine if it contains the second string. The first argument, therefore, is the string to be searched. The second argument is the string to be searched for. The function returns an integer that gives the position in the first string that indicates the beginning of the second string. If the second string is not found in the first string, a zero is returned. The first argument must be a string and not a string literal. The second argument may be a string, a string literal, or a character.

The SUBSTR Function: This function takes three arguments--a string and two integers. It yields a substring of the first argument, which is a string. The second argument is the starting position of the substring in that string. The third argument is the desired length of the substring. The function returns a string. The first argument must be a string and not a string literal.

The DELETE Function: This function takes three parameters--a string and two integers. It deletes a specified substring within the given string, and returns a string. The function takes the first argument, the string, starting at the position specified by the first integer, and deletes the number of characters specified by the second integer. The first argument must be a string, not a string literal.

The INSERT Function: This function takes three arguments--two strings and an integer. It inserts the second string into the first string, and returns a string. The integer specifies the position in the first string where the second string is to be inserted. The first argument must be a string and not a string literal. The second argument may be a string, a string literal, or a character.

The TRIM Function: This function takes a string as an argument and returns a string. It removes all trailing blanks. The argument must be a string, not a string literal.

The LTRIM Function: This function takes a string as an argument and returns a string. It removes all leading blanks. The argument must be a string, not a string literal.

Here is an example that uses all these functions:

```
VAR
  ST8 : STRING[8];
  ST10 : STRING[10];
  I, J, K : INTEGER;
BEGIN
  ST10 := 'ABCDEF';
  I := LENGTH(ST10);  {I equals 6}
  ST8 := 'CDE';
  I := INDEX(ST10, ST8);  {I equals 3}
  J := INDEX(ST8, ST10);  {J equals 0}
  ST8 := SUBSTR(ST10, 3, 2);  {ST8 equals 'CD'}
  ST8 := DELETE(ST10, 3, 2);  {ST8 equals 'ABEF'}
  ST8 := INSERT(ST8, 'HI', 2);  {ST8 equals 'AHIBEF'}
  ST10 := ' A  B  C  ';  {10 characters}
  ST10 := TRIM(ST10);  {ST10 equals ' A  B  C' - 8 characters}
  ST10 := LTRIM(ST10)  {ST10 = 'A  B  C' - 7 characters}
END.
```

Declaring External Procedures and Functions

At Rev. 19.2, a subprogram can call a procedure or function that is contained in the main program. To do this, use the {$E+} and {$E-} compiler switches around the procedure or function declaration in the main program.

Here is an example of a main program that contains an externally declared procedure:

```
VAR
  A, B, C, D : INTEGER;
{$E+}
PROCEDURE ADD (X : INTEGER Y : INTEGER);
  VAR
    Z : INTEGER;
  BEGIN {add}
    Z := X + Y;
    WRITELN('Sum is ',Z)
  END;
{$E-}
PROCEDURE MULT (P : INTEGER; Q : INTEGER); EXTERN;
BEGIN {main}
  A := 8;
  B := 9;
  ADD(A, B);
  C := 5;
  D := 6;
  MULT(C, D)
END.
```

Here is the external subprogram, that calls the procedure:

```
{$E+}
PROCEDURE ADD(X : INTEGER; Y : INTEGER); EXTERN;
PROCEDURE MULT (I : INTEGER; J : INTEGER);
  VAR
    M : INTEGER;
    K, L : INTEGER;
  BEGIN {mult}
    K := 50;
    L := 60;
    M := I * J;
    WRITELN('Mult is ',M);
    ADD (K, L)     {external procedure called here}
  END;
```

Notice that the procedure is declared again under the {$E+} switch, and that this procedure heading ends with EXTERN.

-NOOPT1 AND -NOOPT3 Compiler Options

Two new compiler options were added at Rev. 19.2, -NOOPT1 and -NOOPT3. These options are defaults to the -OPT1 and -OPT3 options, respectively. -OPT1 optimizes less code and generates less efficient code than -OPTIMIZE, but compilation time is faster than with -OPTIMIZE. -NOOPT1 is the default and does not generate less code. -OPT3 optimizes more code and generates more efficient code than -OPTIMIZE, but compilation time is slower than with -OPTIMIZE. -NOOPT3 is the default and does not generate more code.

Severity 3 Error: Use of PACK and UNPACK

At Rev. 19.2, any attempt to use the PACK or UNPACK procedures generates a severity 3 error and causes your program to fail.

DOCUMENTATION CORRECTIONS

The following corrections apply to the second edition of the Pascal Reference Guide (DOC4303-191).

-RANGE/-NORANGE: The -NORANGE option is the default, not -RANGE. See page 2-11 in the new second edition (POLER #52930).

Identifier Length: As of Rev. 19.1, identifiers are no longer truncated to 8 characters. No error message will appear if an identifier has more than 8 characters, but fewer than 32. An identifier with more than 32 characters will generate a severity 1 error. See pages 4-8 and A-5.

SOFTWARE PROBLEMS FIXED

Sets: The set [x..y] is now built correctly when x is greater than y, in which case, the set is empty.

Set Elements: An error message is now given when an integer that is out of range (less than 0 or greater than 255) is assigned as an element of a set.

A runtime error message is now given when an element is added to a set that is out of valid range—less than 0 or greater than 255 (POLER #47371).

An error message is now given when a character string of length greater than 1 is assigned as an element of a set.

"IN" Operator: The "IN" operator now works correctly when the operand is out of range (less than 0 or greater than 255). False is always returned for an out of range value (POLER #44357).

TYPE Declaration: When a colon is used instead of an equal sign in a TYPE declaration, the compiler now gives a severity 2 error message and recovers from the error.

Variable Parameters: When a procedure or function has as a parameter a procedure or function that is a variable parameter, the "VAR" is ignored and a severity 2 error message given.

Syntax Error: A syntactically incorrect program now gives error messages, instead of failing at compile time with an access violation (POLER #43585).

Undeclared Files: A rewrite of an undeclared file now causes a runtime error message instead of an access violation (POLER #43685).

Large Data Structure: An error message is now given at compile time when a data structure to be allocated is larger than one segment (POLER #46630).

Terminal Output: When a negative real was output to the terminal, the minus sign was sometimes missing. The sign has now been restored (POLER #48205).

TYPE Declarations: When a scalar type was declared incorrectly, an access violation was given. An error message is now given (POLER #51346).

External Declaration: The external declaration of a procedure did not make the procedure an entry point. It was unresolved at load time. This problem has been corrected (POLER #52234).

Subranges: Parentheses around a subrange type now generate an error message instead of an access violation.

-LIST: In the listing generated by the -LIST option, the list of options no longer says "OPTIMIZE-2" when it should say "OPTIMIZE", or "UCASE" when it should say "UPCASE".

Internal or unsupported options no longer appear in the listing file.

RESET: A reset of an undeclared file now generates a runtime error message instead of an access violation. A reset of a non-existent file now detects that the file does not exist and generates a runtime error message instead of an access violation.

PL/I, Subset G (PLIG)

SOFTWARE PROBLEMS FIXED

Picture-type Arrays: The problem with printing out picture-type arrays has been corrected by changing library routine P$WALK (POLERs #37392, 48758).

Big Data Size: The problem with data size bigger than one segment has been corrected by updating the allocator (POLER #43320).

I/O Column Format: The problem with I/O column format has been corrected by updating P$EIN (POLER #43857).

Multiple MIDAS Files: During a close, the second file closed with locked records. This problem has been corrected by updating P$CLOS (POLERs #47365, 40077).

I/O Blank After Quote: The problem with I/O blank after quote has been corrected by updating P$LIN (POLER #47917).

Forms "All Clear": The problem with forms "Clear All" has been corrected by changing library routine P$TER (POLER #48002).

I/O Read EOF: The problem with I/O read EOF has been corrected by updating P$GLIN (POLER #51042).

Aggregate Assignment: The problem with aggregate assignment for fixed bin and float has been corrected by modifying PASS2 (POLER #58304).

Float_bin and Float_dec Precision: The problem with float_bin and float_dec precision has been corrected by modifing DECLARE and PASS2 (POLER #60594).

SOFTWARE PROBLEMS OUTSTANDING

TRIM: The second trim is lost in the following PL1G statement:

```
PUT LIST('xx'!!trim(i,'11'b)!!'xx'!!trim(i,'10'b)!!'xx');
```

Label Prefix: A label prefix on a procedure statement causes an error 32 (compiler error) in DECLARE.

For example:

```
a(1): proc;
end;
```

PMA

NEW FEATURES AND CHANGES

Quadruple Floating-point Arithmetic

New instructions have been added in V-mode and I-mode to support the new quadruple-precision floating-point data type.

New Options

The SEG Pseudo_op now takes either PURE or IMPURE as an argument. If the argument is omitted, PURE is assumed. Furthermore, if existing programs do not have at least 18 spaces between the Pseudo_op SEG and a comment, PMA now treats the comment as an operand and generates an error.

Normalization of Floating-point Numbers

Floating-point numbers are now handled more efficiently.

Opcode Tables

These have been made into Insert files.

Other Changes

The Revision 19.2 Assembly Language Programmer's Guide (PTU2600-104) provides further details on the quadruple-precision floating-point data type and the instructions that manipulate it. This update also contains new instructions for the Prime 850 and numerous corrections to previous documentation.

DOCUMENTATION CORRECTIONS

The following corrections apply to the Assembly Language Programmer's Guide (FDR3059-101).

FLR 0: On page 11-7, under the entries for both ZMVD and ZTRN, FLR 0 and not FLR 1 should be set to the number of characters to move (POLER #35275).

Right Brace Negative Sign: Table 11-1 on page 11-1 gives a left brace symbol ({) as a valid negative sign. The correct negative sign is a right brace symbol (}) (POLER #29999).

C Field and XMV Instruction: Page 11-14 incorrectly shows the C field of the decimal control word used with the XMV instruction. This field is used only with the XAD, XMP, XDV, and XCM instructions (POLER # 29999).

FLD addr: On page 11-19, under the entry for FLD addr, change "Load the double precision number..." to "Load the single precision number..." (POLER #40031).

STPM: On page 11-38, the values of the processor model numbers are incorrect. The correct values can be found in the file PRIMOS>KS>SEG14.PMA (POLER #47391).

LF R/LT R: On page 12-17, under the entry for LF R, change "Set R equal to zero..." to "Set RH equal to zero..." Under the entry for LT R, change "Set R equal to one..." to "Set RH equal to one..." These two instructions set only the high-order 16 bits of the specified register to zero and one, respectively (POLERs #48431, 61802).

Previous corrections are listed in the Revision 19.2 Assembly Language Programmer's Guide.

SOFTWARE PROBLEMS FIXED

Pseudo-op Processor: With SYML on, the CALL pseudo-op now recognizes 32-character names.

External Symbols: Using symbolic names for I-Mode registers no longer destroys external symbols.

Single Quote: A "'" in column 1 now works as documented.

Common Blocks: Common blocks with a size of up to 1 segment may now be declared. Previously, any block declared larger than 32767 words was given a size of 0.

Negative exponents: These are now represented correctly.

ROT: The ROT instruction has been corrected.

Symbol Table: The size of this table has been adjusted.

SOFTWARE PROBLEMS OUTSTANDING

Floating Point: In I-mode, a floating-point number used as a literal without a leading "=" sign is not flagged as an error.

LDX const: When the target of a LDX instruction is the X register, no error is flagged.

Final Comment Lines: Comment lines following the END statement of a PMA program generate an error and cause SEG to reset the top of the procedure segment to 177777 (POLER #46341).

RFTNLIB

NEW FEATURES AND CHANGES

Interludes

The use of interludes between R-mode and V-mode gates and libraries has been changed. When using the insert VLUDE.INS.PMA, load the routine with the V-mode interlude only after loading the R-mode FORTRAN library (FTNLIB or SVCLIB).

A warning message is issued by LOAD if routines containing V-mode interludes are loaded before the R-mode library. Also, programs designed for use under PRIMOS II must use SVCLIB.

VFINLIB

NEW FEATURES AND CHANGES

New Data Type

Runtime support has been added to the FORTRAN library for a new data type, REAL*16, which will be supported only by the F77 compiler. For a detailed explanation, refer to the section of this chapter describing F77 and to the new edition of the FORTRAN 77 Reference Guide (DOC4029-192).

Because of this new data type, PL1G no longer returns an error condition for decimal numbers containing between 14 and 28 digits. However, only decimal numbers of between 7 and 28 digits permit double-precision accuracy.

FORTRAN formatted input now rounds on input to provide support for constant rounding in the compiler.

SOFTWARE PROBLEMS FIXED

Format Fields: F$IO77 now produces the correct number of stars when the format field specified is too small for the number (POLER #30123).

Direct I/O: Problems with accessing records using direct I/O have been corrected (POLERs #30212, 46165, 47429).

Missing Library Routines: The missing library routines F$SCCPWR and F$SCIPWR have been included in the library (POLERs #32056, 34326).

Values Out of Range: DEXP$X now returns consistent errors on values out of range (POLER #34034).

Floating Minus Signs: F$IO77/F$IOFTN now recognize as legal specifiers all cases of floating minus signs in B-format specifiers (POLER #34743).

Performance: Library runtime performance degradations after Rev. 18.2 have been resolved (POLER #40423).

Line Printer Control: The form feed printer control character, "-", now works correctly on a line printer (POLER #41658).

Random Number Generator: The repeat interval from the random number generator has been doubled (POLER #47579).

TODEC: TODEC can now convert "-32768" (POLER #48530).

VRPG

NEW FEATURES AND CHANGES

Default Compiler Options

The VRPG driver program, RPGDF.SAVE, which sets the default options of the VRPG compiler, now resides in VRPG>TOOLS. The System Administrator can change the default options by the command:

```
RESUME RPGDF SYSOVL>RPGDATA [options]
```

See Chapter 14 of the System Administrator's Guide (DOC5037-190) for more information.

SOFTWARE PROBLEMS FIXED

Primary and Secondary Files: In some circumstances, the compiler aborted when no primary file was specified. This has been corrected. Now, if a secondary file is specified, the first one is defaulted to the primary file; otherwise, a severity 3 error is issued (POLERs #58983, 52784).

Support of Lowercase: Lowercase literal strings are now supported in VRPG. Previously, all lowercase characters in a VRPG program were converted to uppercase before processing. This prevented the use of lowercase characters in literals and the use of lowercase characters in editwords (POLER #57137).

Output: An overflow output problem could cause missing header lines or extra header lines. This happened if the overflow line was reached when the other conditioning indicator was on. The extra lines were occurring when an output line was conditioned as in the following example, and the control break occurred while the overflow indicator was on.

```
OOUTP    D   06   OANL1
O        OR        L1
O                                  10 'HEADING'
```

The missing lines occurred when an output line was conditioned as in the following example, and the control break occurs before the overflow line is reached.

```
OOUTP    D   06   L1NOA
O       OR        OA
O                                  10 'HEADING'
```

(POLERs #57195, 55435, 55436).

CHAPTER 4

DATA MANAGEMENT SYSTEMS

DBMS

DOCUMENTATION CORRECTIONS

The following correction applies to the DBMS Data Manipulation Language Reference Guide (DOC5308-190).

ON ERROR: In the first example on page 3-45, the "#" symbol was inadvertently omitted in column 7 of the ON ERROR clause.

The example should read:

```
#     FIND NEXT RECORD EMPLOYEE OF SET DEPT-SET
#          ON ERROR 0307 GO TO LABEL-1.
#     GET.
      .
      .
      .
 LABEL-1.
      DISPLAY 'END OF SET ENCOUNTERED'.
#     CLEAR ERROR.
```

The following corrections apply to the DBMS Data Description Language Reference Guide (DOC5717-181).

Data Item/Data Check Clause: The last paragraph on page 2-22 should read:

> When LIST USING is specified, the data item must be of the type CHARACTER n, where n is the exact number of characters allocated to the data item. The value assigned to the data item must be one of the literals in literal-list. The literals in literal-list must be character strings and must include the exact number of characters specified in the CHARACTER clause.

For example:

```
1 REGISTER-CODE;
  TYPE IS CHARACTER 2;
  CHECK IS LIST USING '01', '03', '05'.
```

Reserved Words: In appendix D, add "ACTUAL" to the list of reserved words for schema DDL (POLER #53196).

SOFTWARE PROBLEMS FIXED

FSUBS: FSUB SYSCOM MAP now shows the correct size of ERITEM (POLER #41450).

DBACP: DBACP now allocates before-image files up to the defined maximum size of INTEGER*4 (POLER #43052).

DBACP RESTORE now produces the proper error message when an invalid treename for an after-image file is entered (POLERs #32638, 32211).

DBACP SAVE TO TAPE function has been modified to:

1. Prompt the user to mount new tape when a tape error occurs;
2. Ask for the proper mag tape unit number if one is not assigned or online;
3. Properly unlock the data base for other users when a SAVE procedure is aborted before it reaches completion (POLERs #34481, 35201, 40760, 36464, 43922, 58278).

DBACP PACK AREA now detects a corrupted data base and damages it no further. This modification also increases performance by a minimum of 60 per cent (POLERs #32867, 32869).

DBACP now closes files consistently, even files allocated on a second volume (POLERs #34772, 36602, 37900, 37752, 81962, 34052).

DBACP recognizes a maximum of 16 volumes during file allocation. Any volume with LDN>16 is not recognized as valid.

DMLCP: After a REMOVE or DELETE, set privacy locks are properly maintained (POLER #44232).

The STORE, FIND, and DELETE commands now treat data type conversions in a consistent manner (POLERs #47676, 47681).

DBMS users no longer occasionally hang on an EXIT DBMS call (POLER #56262).

DBUTL: The DBUTL commands RDIR and ADIR now show in parentheses the correct octal equivalents of the decimal addresses (POLER #29404).

DBUTL now reads correct nodes while verifying sets (POLERs #47967, 47968, 55683).

SCHDEC: SCHDEC now properly handles output source file names up to 80 characters long (POLER #36005).

SCHDEC now produces the correct output line for a picture clause of the form "V99" (POLER #33847).

SCHDEC now accepts single quotes around a treename with a password as the output file name (POLER #33119).

When a schema is decompiled with SCHDEC, a signed item (e.g., PIC S99V99) appears as unsigned (PIC 99V99). SCHEMA treats signed and unsigned fields alike (POLER #42934).

SCHED: SCHED produces the error message "ONLY CHECK AND TYPE CODE CLAUSE MAY BE CHANGED" when a user tries to change an item in any other way (POLER #29403).

SCHED checks the length of the SEARCH/SORT/CALC key. If its length exceeds the defined maximum of 30 words, SCHED produces the message "SEARCH/SORT/CALC KEY LENGTH GREATER THAN 30 WORDS" (POLER #43242).

After a successful session of SCHED, after-imaging is turned off, if it was on. The user is informed of the fact by the message "AFTER-IMAGING IS SHUT OFF" (POLER #48540).

SCHED now gives statistics of up to 20 buckets of fragmented records.

SCHEMA: SCHEMA checks the length of the SEARCH/SORT/CALC key. If its length exceeds the defined maximum of 30 words, SCHEMA produces the message "SEARCH/SORT/CALC KEY LENGTH GREATER THAN 30 WORDS" (POLER #43242).

SOFTWARE PROBLEMS OUTSTANDING

DBACP: The DELETE SCHEMA command sometimes causes DBMS errors because the SD# entry was not deleted (POLERs #37900, 46653, 81975).

Gives uninformative error messages when it fails to open DB# file (POLER #33542).

EXPAND of areas may cause high I/O usage (POLER #32867).

Does not always correctly restore schemas saved without the after-image file (POLERs #34679, 25408, 48007).

ALLOCATE of an empty area gives 33 EOF error (POLER #46238).

Bad linkage of available nodes causes fatal internal DBMS error (POLER #37505).

When allocating a very large database, DBACP returns negative numbers for file sizes (POLER #58656).

DBACP EXPAND gives unpredictable results (POLERs #41637, 44226).

DBACP EXPAND gets error code 31 on trying to open a SEGDIR file (POLER #43624).

DBUTL: DBUTL documentation and the HELP option state incorrectly that both major and minor codes are given for the MON command (POLER #47969).

DMLCP: DMLCP fails to detect an area opened twice at runtime. "Close all areas" closes only one file unit. CLUP closes the second file unit, but issues the message "NO OPENED AREAS FOUND" (POLER #47675).

It is possible to initialize and store non-numeric values for calc key fields of type PIC 9 (POLER #47677).

DMLCP does not return an error on an illegal store using a variable length record (POLER #41892).

Conversion between REAL*4 and packed decimal as data moves between schema and subschema is not correct. The data changes without updates (POLER #40392).

An internal fatal error results when you attempt to store a record, with no duplicates allowed, for the second time (POLER #52834).

RLIB: The transaction bit map may overflow if there is a hung or very slow user (POLER #29298).

ENVIRONMENT

Rev. 19.2 DBMS requires PRIMOS Rev. 19.1 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.

DBMS/QUERY

DOCUMENTATION CORRECTIONS

The following corrections apply to the DBMS/QUERY Reference Guide (IDR4607-182).

CREATE FORMAT: On page 10-3, in the last paragraph under the CREATE FORMAT command, delete the phrase "If there are no errors..." The sentence should read simply: "QUERY catalogs the format."

EDIT FORMAT: On page 10-4, under the EDIT FORMAT command, the next to last paragraph erroneously implies that only a format that is syntactically correct will be cataloged. In fact, whenever you issue the FILE command, the format is cataloged. If any errors are present, QUERY returns you to the editor to correct them. You can also choose to delay the editing because the format remains cataloged.

If you use the QUIT command rather than FILE, any changes made during the current editing session will not be saved. If you QUIT from the initial creation of the format, no cataloging occurs.

SOFTWARE PROBLEMS FIXED

Line Truncation: When argument expansion in a procedure causes a line to exceed 160 characters, DBMS/QUERY no longer truncates the line to 160 characters.

Command Display: If procedure echoing is enabled and a command is split over more than one line (either by the user with the tilde character or by the argument expander), the entire command is displayed, instead of just the first line.

PICTURE Information: When displaying a PICTURE string from a COBOL subschema, picture information following a 9(x) is no longer deleted when x is greater than 4. This problem occurred in the DESCRIBE RECORDS and SAVE TABLE commands. In the SAVE TABLE command, the incorrect description was written out to the table description file, which resulted in the failure of application programs that used the file as an INSERT or INCLUDE file (POLER #47680).

SELECT: When selecting records from a database with a SELECT command and a compound WHERE clause--that is,

SELECT <item-list> FROM RECORD <record-name> WHERE <item-name1> = '<string1>' AND <item-name2> = '<string2>'

DBMS/QUERY sometimes crashed with a UII$ error condition after retrieving at least one virtual record. This problem has been corrected (POLERs #41337, 44925, 51498, 59313).

SOFTWARE PROBLEMS OUTSTANDING

No more than 15 areas may be open at a time (POLER #60201).

ENVIRONMENT

Rev. 19.2 DBMS/QUERY requires Rev. 19.2 DBMS and Rev. 19.2 PRIMOS.

MIDAS

DOCUMENTATION CORRECTIONS

The following correction applies to the Midas User's Guide (IDR4558-176).

GDATA$ Calling Sequence: At the bottom of page 6-47, under the argument status, add the value of +1 for EOF (POLER #29998).

SOFTWARE PROBLEMS FIXED

File Allocation: MIDAS no longer erroneously extends a direct access file allocation (POLER #41451).

Closing Subfiles: BILD$R now closes subfiles when the end of processing is indicated and no records have been added (POLER #47488).

KBUILD: KBUILD now accepts output file pathnames longer than 40 characters (POLER #51154).

At segment boundaries, KBUILD adds direct access records correctly (POLER #60202).

MPACK: The MPACK "UNLOCK" option now releases access controls after use (POLER #58926).

MDUMP: MDUMP now dumps direct access records (POLER #60319).

SOFTWARE PROBLEMS OUTSTANDING

"DISK FULL" Error: MIDAS leaves a file in an inconsistent state when a "DISK FULL" error arises on a call to the file system. MIDAS aborts the process.

"ALL FILE UNITS IN USE" Error: Receiving the "ALL FILE UNITS IN USE" error on a call to the file system, MIDAS is unable to multiplex its file unit usage and aborts the process.

ENVIRONMENT

MIDAS Rev. 19.2 is compatible with other Rev. 19.2 products. This version of MIDAS is intended for Revs 18.5, 19.1, and 19.2 COBOL, BASICV, and POWERPLUS. It should not be installed on the same system as MIDASPLUS. While MIDAS Rev. 19.2 should be able to run against PRIMOS Revs 18.3 and later, it is not supported for those releases. OAS prior to Rev. 3.0 should not be used with this release of MIDAS.

MIDASPLUS

NEW FEATURES AND CHANGES

Open and Close

Since PRIMOS Revs 18.5 and 19.1, file opening and closing requirements for MIDAS have been identical to those for MIDASPLUS.

MIDASPLUS requires a call to OPENM$ or NTFYM$ before a file can be accessed via the online MIDASPLUS routines. Without the call, MIDASPLUS error code 23 is returned from the online routines.

Previously, MIDAS did not require a call to OPENM$ or NTFYM$ before a file could be accessed via the MIDAS online routines. You could therefore open a file with SRCH$$ and then access it directly with an online routine like FIND$. Although this action caused MIDAS to operate inefficiently, it would still run.

Opening a file for use by BILD$R, PRIBLD or SECBLD has not changed. In these cases, the file must be opened through a PRIMOS call and must not use OPENM$ or NTFYM$.

MIDAS closes the file specified in a call to CLOSM$ even if the file is not a MIDAS file (known to MIDAS via a call to OPENM$ or NTFYM$). This is not the case with MIDASPLUS. Via calls to CLOSM$, MIDASPLUS closes only those files which have been opened as MIDASPLUS files by calls to OPENM$ or NTFYM$.

When MIDASPLUS files are opened by SRCH$$ or TSRC$$, the call to NTFYM$ must be made after the open. When MIDASPLUS files are closed by SRCH$$ or TSRC$$, the call to NTFYM$ must be made before the close.

Counts of Entries Added and Deleted

In order to increase performance, MIDASPLUS does not write to the MIDAS file the updated values of "entries added" and "entries deleted" every time an entry is added or deleted. Instead, these counters are updated in memory and written to the disk when the last user closes the file (by a CLOSM$ or a NTFYM$ call or MPLUSCLUP or automatic cleanup by the static on-unit).

Should the last user fail to close the file, the counters in the file will not be correctly updated. In such a situation, the counters will always overestimate the number of entries in the file by between 1 and 100 entries. The counters can be corrected by running MPACK on the file. Bear this in mind when using CREATK to display the counter values while the file is in use by MIDASPLUS users.

New Error Codes

Some new error codes have been added to MIDASPLUS. The meaning of these codes is described below.

Error Code	Meaning
28	Attempt to write to read-only file.
40	Fatal internal error within MIDASPLUS.
41	Timeout occurred while attempting to get a buffer.
90-92	Network errors.
'10001	Error in close.
'10002	File unit table or shared file table is full.
'10004	Fatal internal error (from OPENM$ only).
'10005	Rev. level regression.
'10006	File in use by unshared MIDAS.
'10007	File in use by MIDASPLUS.
'10008	File is being used by a MIDAS utility or user calls to PRIBLD, SECBLD or BILD$R.
-4	Fatal internal error (from GDATA$ only).
19	Full disk condition occurred during add operation.

Fatal internal errors indicate that an internal error within MIDASPLUS has been detected. These errors are returned only when there is no other error code to identify the problem; the user should contact Prime Customer Service.

MIDASPLUS and OAS

Revisions of Prime's Office Automation System (OAS) prior to Rev. 3.0 will not run with MIDASPLUS.

DOCUMENTATION CORRECTIONS

The following correction applies to MIDASPLUS (PTU2600-098).

MDUMP: On page 98-17, under the entry MDUMP's Uses, change the sentence "You can dump a MIDAS file, edit the resulting sequential file, and feed the edited file to the KBUILD utility" to "...edit the resulting sequential file if the data is in ASCII format, and feed..." The editor will corrupt data in Comp or Comp-3 format (POLER #60630).

SOFTWARE PROBLEMS FIXED

Closing Subfiles: BILD$R now closes subfiles when the end. of processing is indicated and no records have been added (POLER #47488).

KBUILD: KBUILD now accepts output file pathnames longer than 40 characters (POLER #51154).

At segment boundaries, KBUILD adds direct access records correctly (POLER #60202).

Opening Remote and Local Files: Previously, if a remote file were opened after a local file, attempts to access the remote file could fail with a MIDASPLUS 23 error. This no longer happens (POLER #58712).

MSGCTL: MSGCTL no longer turns the debug option off regardless of the flag values passed to it.

Direct Access Deletes: Direct access calls to DELET$ no longer fail in internal calls to LOCK.

MPACK: The MPACK "UNLOCK" option now releases access controls after use (POLER #58296).

MDUMP: MDUMP now dumps direct access records (POLER #60319).

MPLUSLB.BUILD.CPL: The force load flag in the build MPLUSLB.BUILD.CPL has been reset so that the whole library is not loaded with each user's program (POLER #60694).

SOFTWARE PROBLEMS OUTSTANDING

Read-only Files: Read-only access is not available at present; thus, MIDASPLUS does not support read-only MIDAS files. This problem derives from the protection against concurrent usage by MIDASPLUS and unshared MIDAS. MIDASPLUS attempts to write to subfile 0 (POLER #60318).

CREATK: CREATK index entry counts are not always correct (POLER #47163).

Timing Out: If two applications are simultaneously opening the same file, they may both time out trying to get an internal lock (POLER #56382).

MDUMP: MDUMP puts the key in the middle of very long records (POLER #57851).

ENVIRONMENT

MIDASPLUS Rev. 19.2 depends on:

BASICV	18.5, 19.1 or 19.2
COBOL	18.5, 19.1 or 19.2
POWERPLUS	18.5, 19.1 or 19.2
PRIMOS	19.1 or 19.2 (will not run prior to 19.1)

POWERPLUS

NEW FEATURES AND CHANGES

Converting to the New PROC Format

This enhancement requires a change in the structure of the procedure dictionary file. Installation of Rev. 19.2 POWER necessitates running a utility to convert POWR##. Existing procedure files will not have associated login ids and will all be marked as public files. The utility to convert to the new PROC format is in POWERPLUS>TOOLS and can be run by the POWERPLUS install command file.

DOCUMENTATION CORRECTIONS

The following correction applies to the PRIME/POWER Guide (PDR3709-173).

POWER EDITOR: The POWER EDITOR can edit PROCEDURE files of up to 950 lines. It cannot edit PROCEDURE files of up to 1200 lines, as stated on page 15-3 (POLER #51447).

SOFTWARE PROBLEMS FIXED

FIND: FIND <search-expression> AND <search-expression> now works properly with tables (POLER #29703).

FIND with OR on the MIDAS primary key now works correctly (POLER #44928).

Descriptors from Linked Fields: Descriptors from linked fields now appear on reports with more than two rows (POLER #41468).

DUMP: DUMP of a set no longer gives bad data in the date field from a linked file that is 2 links away from the current file (POLER #41994).

Error Messages: If the field length is defaulted in a CREATE CHANGE, a "DESCRIPTOR OUT OF RANGE" message is displayed (POLER #43565).

MIDASPLUS no longer gives the error message "UNIT NOT OPEN" when POWERPLUS is used with linked files. This also allows the OAS user to run correctly after a POWERPLUS session with linked files (POLER #60775).

COMBINE: COMBINE on two disjoint large sets no longer produces a non-empty set (POLER #45887).

LWORD: The temporary change to a user's lword to suppress password echoing could cause a forced logout over dial-in lines. This problem has been corrected (POLER #45926).

Character Position>5000 in a TABLE: POWER now allows a TABLE to start at character position>5000 when the record is sufficiently large (POLER #47925).

PRINT -AT DS.ONE: PRINT -AT DS.ONE no longer results in printer DS and FORM ONE (POLER #52199).

Detection of Illegal Numeric Output Format: An illegal numeric output format is now detected during the creation of a new form (POLER #27496).

More than One Function per Expression: The user is now able to use more than one POWER function in an expression. Previously, for example,

```
N1 = SUM(desc1)*AVG(desc2)
```

would result in N1 equaling only SUM(desc1) (POLER #40135).

Display: The DISPLAY USING REPORT command no longer causes the user's terminal to hang when the report contains fields from linked files (POLER #51051).

Page number and column headings are no longer scrolled off the screen during a display (POLER #58301).

Reports are now able to display lines containing no descriptors (POLER #60776).

After an error in ADD USING SCREEN on a terminal other than a PT45, POWER no longer scrolls up two rows (POLER #61870).

In a report, correct subtotals are now displayed for the 9th descriptor with subtotals (POLER #45141).

Improved Validation: Table descriptors are now validated correctly during a non-screen ADD (POLER #60623).

Validation of descriptors no longer causes MIDASPLUS errors.

POWER*: Code now attaches to the partition with POWER* when a terminal type is added or the validity of a terminal type is checked for a screen function.

CLOSM$: The CLOSM$ call has been changed to CLOSE for non-MIDAS files.

ENVIRONMENT

Rev. 19.2 PRIME/POWER must be installed with Revs 19.1 or 19.2 PRIMOS, FORTRAN, and MIDAS/MIDASPLUS.

CHAPTER 5

HARDWARE SUPPORT

THE PRIME 9950

INTRODUCTION

Rev. 19.2 provides support of a new processor, the Prime 9950. Specific changes for the 9950 include software support of the following features.

TIME-OF-DAY CLOCK

The 9950 has a diagnostic processor with a battery-backed time-of-day clock. PRIMOS has been changed to read the clock during system cold start and warm start and to set the date and time of the system to the nearest 4-second interval.

On the Prime 9950, the system clock can be set or reset automatically without operator intervention during system cold start and warm start.

To use the automatic time-setting feature of the 9950, delete the SETIME command from existing C_PRMO (or PRIMOS.COMI) files.

LARGER PHYSICAL MEMORY

Support has been added to PRIMOS for physical memory of up to 16 megabytes for the Prime 9950. The previous limit was 8 megabytes.

ENVIRONMENTAL SENSORS

Changes have been made to PRIMOS to support machine checks due to environmental sensors. The Prime 9950 has sensors in the hardware to detect inadequate air flow in the cabinet, overheating of the cabinet, and low battery conditions.

When such problems occur, the diagnostic processor indicates the condition by causing a machine check. PRIMOS responds to this machine check by disallowing supervisor console input, signaling the logout condition for all users, waiting for all users to be logged out, and then shutting down the system.

ICS2 COMMUNICATIONS CONTROLLER

INTRODUCTION

The ICS2 is a new communications controller, which controls up to 64 asynchronous lines, in increments of 4 lines. At cold and warm starts, software is downline-loaded into the ICS2 in order for it to operate.

In general, the ICS2 controller is used similarly to the ICS1 controller. New commands and directives introduced at Rev. 19.1 for the ICS1 also apply to the ICS2.

For details on ICS1 and ICS2 controllers, see

- The Revision 19.1 Software Release Document (MRU4304-009), which summarizes changes made at Rev. 19.1 for the ICS1.
- The System Administrator's Guide Update, Rev. 19.1 (UPD5037-191)
- The System Administrator's Guide Update, Rev. 19.2 (UPD5037-192)

In particular, the updates to the System Administrator's Guide contain information on configuring ICS controllers, and some additional guidelines for their use.

STATUS

The STATUS COMM command now recognizes ICS2 controllers and displays appropriate information for each one present in the system. For example:

```
OK, STATUS COMM

                        Device      Lines
 Controller   Type      Address   Async Sync

 ICS2                     11        32    0
 AMLC         DMQ         54        16    0

OK,
```

USAGE

The parameters %ASYNC and %ICS now indicate CPU percentage required to support ICS1 and ICS2 controllers.

PRIMOS COLD START

Downline Load File

Each ICS2 has to be downline loaded at cold start. The code and data is contained in the file ICS2.DL, which must be located in UFD DOWN_LINE_LOAD*.

The New ICS INPQSZ Configuration Directive

A new cold start directive has been introduced at Rev. 19.2, to provide functionality similar to the AMLIBL directive for AMLCs. The ICS INPQSZ directive changes the size of the ICS-to-Prime input queues from the default value of 63 (octal 77). The sizes of all ICS input queues (including ICS1 input queues) are changed by this directive.

The format of the directive is:

▶ ICS INPQSZ n

where n is the length of the queue, specified in octal. The first octal digit of n must be 1, 3, or 7. The remaining digits (up to 4) must all be 7. These guidelines ensure that n is one less than a power of 2, which is required. Examples of possible values for n in octal are 177, 377, and 777.

This directive may be necessary to avoid losing ICS2-to-PRIMOS data when the amount and rate of input data is high, e.g., when several terminals are doing page transmissions. For further details, see the System Administrator's Guide Update, Rev. 19.2 (UPD5307-192).

PRIMOS WARM START

At Rev. 19.1, warm start of a system using ICS1 controllers took longer than did warm starts at earlier revs. At Rev. 19.2, ICS2 controllers also cause a longer warm start.

WARNING MESSAGES

The following messages, which can be printed out during cold start and warm start, are warnings only. They indicate hardware failure that affects only some of the lines on a particular controller. The system will continue to cold start, but the indicated lines will not be available.

ICS2 async line #nn on device dd is inoperable.

ICS2 device dd has an inoperable line card in slot #nn.

ASYNCHRONOUS LOGICAL LINE NUMBER ASSIGNMENT

In a system that includes AMLC, ICS1, and ICS2 controllers, logical line numbers are assigned much as they were at Rev. 19.1.

When all AMLCs have been allocated line numbers, ICS1 and ICS2 line numbers are then assigned starting at the next 16 ('20) line boundary. Line numbers for each ICS controller always begin at a modulo-8 value. ICS controllers are assigned line numbers in the order at which the controllers appear in the table displayed by the STATUS COMM command.

For further discussion of the ICS2 and configuration of terminals attached to it, see the System Administrator's Guide Update, Rev. 19.2 (UPD5037-192).

PST 100 TERMINAL

FUSES

This information updates the PST 100 Installation and Fault Isolation Guide, (DOC6987-001).

Note

The kit supplied with each PST 100 contains two fuses.

- One is for domestic (60 Hz) units. It is .250" x 1.250" (6.35 mm x 31.75 mm).
- One is for international (50 Hz) units. It is .196" x .787" (5 mm x 20mm).

These fuses are not interchangeable. A defective fuse must be replaced with one that is the same size physically as well as electrically. If the incorrect fuse is used, the terminal may not function.

USE AS A SUPERVISOR TERMINAL

If the PST 100 terminal is used as a supervisor terminal (console) for a Prime 50 Series computer, then the connecting cable CBL3692-90X must be REV J or greater. If the cable is an earlier REV, then pins 4 and 5 of connector J-3 must be jumpered together.

The updates the PST 100 Installation and Fault Isolation Guide (DOC6987-001).

PST 100 AUXILIARY PORT OPERATION

The PST 100 terminal includes an auxiliary port for the connection of serial devices. This port allows transfer of data from the terminal to another device for the purpose of copying either data on the PST 100 screen or data received from the host computer connected at the PST 100 main serial port. The auxiliary port is not a printer port as no special printer protocol is used by the PST 100 when transferring data.

AUX SEND key

The PST 100 operator would normally initiate data transfer to the auxiliary port by using the AUX SEND key. This key can cause two types of aux send actions depending on what other key is augmented with (it also is used to control the System/Status line when augmented with the CONTROL key). The action of the key may be affected by whether the terminal is in Character Mode or not. When in Local or Block Mode, the action of the AUX SEND key is generated internally with no interaction with the host computer. When in Character Mode however, the AUX SEND key will only perform its specified function if the host computer echos back the character that the AUX SEND key causes to be transmitted.

The two types of transmission caused by the AUX SEND key are:

- PRINT SCREEN: Pressing AUX SEND either unaugmented,. or augmented only with LOCK, causes the characters on the screen to be transmitted exactly as shown with no visual attributes and no separation between lines. In other words, exactly 1920 characters (3840 if in Two Page Mode) will be transmitted with no separation by control codes such as carriage return, line feed, etc.

- AUXILIARY SEND: Pressing AUX SEND augmented with SHIFT also causes 1920 characters (3840 if in Two Page Mode) to be transmitted with no separation by control codes. If Logical Attribute Mode is reset (that is, no logical attributes are in force), the result is exactly the same as a PRINT SCREEN described above. If Logical Attributes Mode is set, the data transmitted is determined by the settings of Selected Area Transfer Mode and Unprotected/Modified Mode:

  With Logical Attributes Mode set, and Selected Area Transfer Mode reset, only characters within selected areas are transmitted as displayed on the screen; all other characters are transmitted as spaces. When Selected Area Transfer Mode is reset, the setting of Unprotected/Modified Mode is ignored.

  With Logical Attributes Mode set, Selected Area Transfer Mode set, and Unprotected/Modified Mode reset, only characters within unprotected areas are transmitted as dislayed on the screen; all other characters are transmitted as spaces.

  With Logical Attributes Mode set, Selected Area Transfer Mode set, and Unprotected/Modified Mode set, only characters within modified areas are transmitted as displayed on the screen; all other characters are transmitted as spaces.

Media Copy Command

The Media Copy command allows the host computer to initiate auxiliary port operations. There are four different parameters for the Media

Copy command, two of which are identical to the character strings generated by the AUX SEND key when in Character Mode. The different Media Copy operations are as follows:

- "ESC [0 i": This escape sequence initiates a PRINT SCREEN operation as described above. Pressing the AUX SEND key when in Character Mode causes this sequence to be sent to the host.

- "ESC [5 i": This escape sequence causes data received at the main serial port to be copied to the auxiliary port exactly as received at the terminal. The data received is not displayed on the terminal screen.

- "ESC [4 i": This escape sequence turns off copying of data received at the main serial port to the auxiliary port and causes this data to again be displayed on the terminal screen.

- "ESC [>0 i": This escape sequence initiates an AUXILIARY SEND operation as described above. Pressing the AUX SEND key augumented with SHIFT when in Character Mode causes this sequence to be sent to the host.

FRENCH VERSION OF PST 100

The following is a description of the difference between the standard PST 100 terminal and the French version of the PST 100.

Keyboard

The French keyboard is an AZERTY layout instead of the U.S. standard QWERTY. In addition, many non-alphabetic keys have been moved from their postions on the U.S. keyboard. (See the keyboard layout in Figure 5-1.)

Character Set

Characters that are common to both the French and U.S. character set will appear the same with the exception of the ^ accent. In the French character set, this symbol is smaller and higher up in the character cell so as to fit above vowels when it is used as an accent in dead key action.

In the French character set the [character (left bracket) which is heavily used in PST 100 escape sequence does not exist. Its position in the ASCII table has been replaced with an underlined o symbol. Thus, when generating escape sequences from a French keyboard, this symbol (which is a shifted right parenthesis) should be substituted for the left bracket.

When using a French keyboard with the standard U.S. character set, some unexpected results may occur because the French character set uses a modified ASCII table in which some French characters replace standard ASCII characters. (See Table 5-1.)

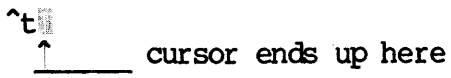
Functionality Changes

The French version of the PST 100 has the ability to handle "dead keys". These are accent marks that when received by the terminal either from the keyboard or from the host result in the cursor not moving to the next position as would normally happen. When a dead key is followed by a lower case vowel key, the accent and the letter are combined in one screen position and the cursor is moved to the next position.

The two accent marks that cause dead key action are umlaut and the ^ symbol. These are both on the same key, located to the right of the letter P. When either of these accents is followed by a lower case a,e,i,o,u the dead key action as described in the previous paragraph will occur, resulting in an accented letter. Thus, the following are all the possible legal combinations of the dead keys and characters:

ä ë ï ö ü â ê î ô û

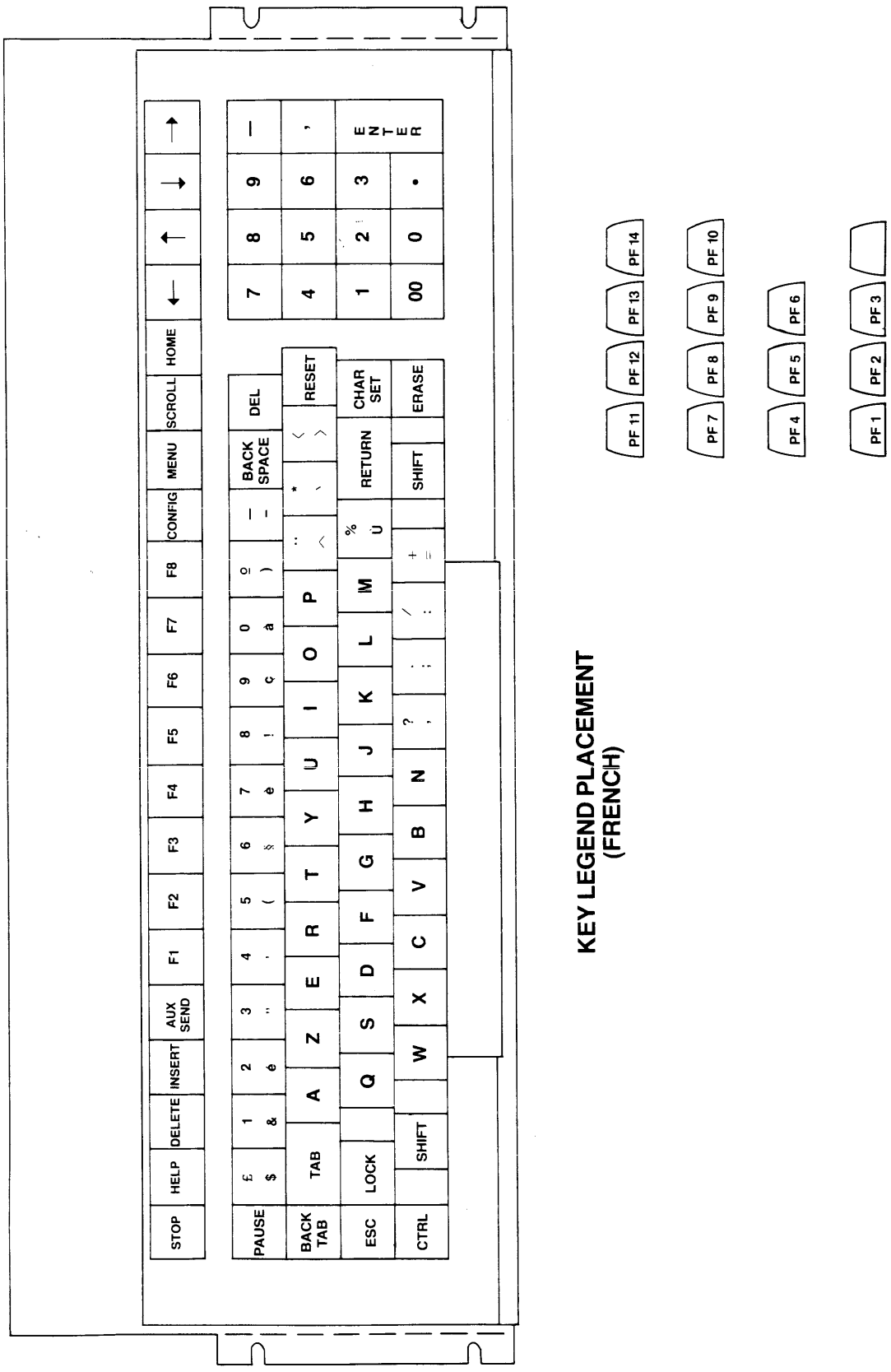
When one of these accents is followed by any character other than lower case a,e,i,o,u, the result will be the accent mark in its original position, followed by the next character received in the first available position to the right of the accent mark. The cursor will end up on the next available position after the most recent character. For example, if the terminal receives ^ followed by t, the result would be:



The action of the characters umlaut and ^ are true whether in character mode or block mode and whether the characters come directly from the keyboard or from the host.

PST 100 Transmission of Dead Keys

Valid dead key combinations that appear on the PST 100 screen are transmitted to the host in the same manner in which they were originally received. That is, the dead key combination is sent as two separate characters, the ASCII code for th eaccent mark followed by the ASCII code for the vowel under the accent. There are no special ASCII codes defined for the character resulting from the combination of the accent marks umlaut or ^ and the vowels.



PST 100 French Keyboard
Figure 5-1

Table 5-1
PST 100 ASCII Character Set (French)

PST 100 French Character Set

	000	001	010	011	100	101	110	111	1010	1011	1110	1111
0000	NUL	DLE	SP	0	à	P		p				
0001	SOH	DC1	!	1	A	Q	a	q	â		ä	
0010	STX	DC2	"	2	B	R	b	r				
0011	ETX	DC3	£	3	C	S	c	s				
0100	EOT	DC4	$	4	D	T	d	t				
0101	ENQ	NAK	%	5	E	U	e	u	ê	û	ë	ü
0110	ACK	SYN	&	6	F	V	f	v				
0111	BEL	ETB	'	7	G	W	g	w				
1000	BS	CAN	(	8	H	X	h	x				
1001	HT	EM	)	9	I	Y	i	y	î		ï	
1010	LF	SUB	*	:	J	Z	j	z				
1011	VT	ESC	+	;	K	°	k	é				
1100	FF	FS	,	<	L	ç	l	ù				
1101	CR	GS	-	=	M	§	m	è				
1110	SO	RS	.	>	N	^	n	¨				
1111	SI	US	/	?	O	_	o	DEL	ô		ö	

Notes

1. Columns 000 to 111 contain the character set used both internally and externally (on the communication line).

2. Columns 1010 to 1111 are codes which are internal to the PST 100 only.

READER RESPONSE FORM

Your feedback will help us continue to improve the quality, accuracy, and organization of our user publications.

1. How do you rate the document for overall usefulness?

   ___excellent ___very good ___good ___fair ___poor

2. Please rate the document in the following areas:

   Readability: ___hard to understand ___average ___very clear

   Technical level: ___too simple ___about right ___too technical

   Technical accuracy: ___poor ___average ___very good

   Examples: ___too many ___about right ___too few

   Illustrations: ___too many ___about right ___too few

3. What features did you find most useful? _____

   _____

   _____

   _____

4. What faults or errors gave you problems? _____

   _____

   _____

   _____

Would you like to be on a mailing list for Prime's current documentation catalog and ordering information? ___yes ___no

Name: _____ Position: _____

Company: _____

Address: _____

_____ Zip: _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

First Class Permit #531 Natick, Massachusetts 01760

BUSINESS REPLY MAIL

Postage will be paid by:

PRIME

Attention: Technical Publications
Bldg 10B
Prime Park, Natick, Ma. 01760