Shared Resource Management
HP Series 200 Workstation Manual

Manual Part No. 98619-90051

© Copyright 1984, Hewlett-Packard Company.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another language without the prior written consent of Hewlett-Packard Company. The information contained in this document is subject to change without notice.

Use of this manual and flexible disc(s) or tape cartridge(s) supplied for this pack is restricted to this product only. Additional copies of the programs can be made for security and back-up purposes only. Resale of the programs in their present form or with alterations, is expressly prohibited.

Restricted Rights Legend
Use, duplication, or disclosure by the Government is subject to restrictions as set forth in paragraph (b)(3)(B) of the Rights in Technical Data and Software clause in DAR 7-104.9(a).

Hewlett-Packard Company
3404 East Harmony Road, Fort Collins, Colorado 80525

Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

May 1984...First Edition

Warranty Statement

Hewlett-Packard products are warranted against defects in materials and workmanship. For Hewlett-Packard computer system products sold in the U.S.A. and Canada, this warranty applies for ninety (90) days from the date of shipment.* Hewlett-Packard will, at its option, repair or replace equipment which proves to be defective during the warranty period. This warranty includes labor, parts, and surface travel costs, if any. Equipment returned to Hewlett-Packard for repair must be shipped freight prepaid. Repairs necessitated by misuse of the equipment, or by hardware, software, or interfacing not provided by Hewlett-Packard are not covered by this warranty.

HP warrants that its software and firmware designated by HP for use with a CPU will execute its programming instructions when properly installed on that CPU. HP does not warrant that the operation of the CPU, software, or firmware will be uninterrupted or error free.

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. HEWLETT-PACKARD SHALL NOT BE LIABLE FOR CONSEQUENTIAL DAMAGES.

HP 9000 Series 200
For the HP 9000 Series 200 family, the following special requirements apply. The Model 216 computer comes with a 90-day, Return-to-HP warranty during which time HP will repair your Model 216, however, the computer must be shipped to an HP Repair Center.

All other Series 200 computers come with a 90-Day On-Site warranty during which time HP will travel to your site and repair any defects. The following minimum configuration of equipment is necessary to run the appropriate HP diagnostic programs: 1) ½ Mbyte RAM; 2) HP-compatible 3½" or 5¼" disc drive for loading system functional tests, or a system install device for HP-UX installations; 3) system console consisting of a keyboard and video display to allow interaction with the CPU and to report the results of the diagnostics.

To order or to obtain additional information on HP support services and service contracts, call the HP Support Services Telemarketing Center at (800) 835-4747 or your local HP Sales and Support office.

* For other countries, contact your local Sales and Support Office to determine warranty terms.

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Table of Contents

Introduction

About Revision Numbers ... vii
Additional References ... vii
This Manual's Contents ... viii

Chapter 1: HP Series 200 BASIC Workstation Use on SRM ... 1
System Concepts ... 1
Shared Resource Support of the BASIC Language ... 1
SRM's Hierarchical Directory Structure ... 2
Uses of the Hierarchy: An Example ... 2
Capabilities of Directories ... 3
Referring to Directories and Files in the Hierarchy ... 3
How the SRM System Stores Remote Directories and Files ... 4
Non-Contiguous Storage of Remote Files ... 4
Space Allocation for Remote Directories and Files ... 4
Shared Access to Remote Directories and Files ... 4
Controlled Access: Password Protection ... 4
Exclusive Access: Locking Files ... 5
How the SRM System Manages Shared Peripheral Use ... 5
Using Your BASIC Workstation on SRM ... 6
Booting From the SRM ... 6
Selecting an Operating System ... 7
Automatic Configuration ... 8
Accessing the Shared Mass Storage Device ... 8
Creating Directories and Files ... 9
Creating Directories ... 9
Creating Files and Other Directories Under a Directory ... 10
Copying Files ... 12
Using the COPY Statement ... 12
Other Uses of COPY ... 13
Using LOAD and STORE ... 13
Copying Item-by-Item Using ENTER and OUTPUT ... 13
Using a Shared Printer or Plotter ... 14
Spooling Using PRINTER IS and PLOTTER IS ... 14
Writing Files to the Spooler Directories ... 15
Sending Program Output to a Shared Printer ... 15
Appearance of Output ... 16
Preparing Plotters ... 16
Aborting Printing/Plotting In Progress ... 16
Protecting Files and Directories ... 16
Specifying Passwords ... 18
Purging Remote Files and Directories ... 19
Accessing Files Created on Non-Series 200 SRM Workstations ... 19
Locking and Unlocking Remote Files ... 20
Returning to Local Mass Storage ... 20

Modifying Existing Programs to Access Shared Resources 21
Files Specifiers 21
Composition of Files Names 21
File and Mass Storage Device Specification in String Variables 21
Mass Storage Unit Specification 22
Allowing for Directory Paths 22
Passwords and Protect Codes 23
BASIC Language Reference for HP Series 200 Workstations 25
Syntax for Remote File and Directory Specification 26
Remote File Specifier 26
Directory Path 27
Remote msus 28
Directory Specifier 30
Access Capability Requirements 31
Table of Access Capabilities Required for Keyword Use 32
Using Protected Files Created on a Pascal Workstation 33
Summary of BASIC Keyword Use on SRM 34
ASSIGN 35
CAT 36
CHECKREAD 40
CONTROL 41
COPY 42
CREATE ASCII 43
CREATE BDAT 44
CREATE DIR 45
ENTER 46
GET 47
INITIALIZE 48
LOAD 49
LOADSUB 50
LOCK 51
MASS STORAGE IS (MSI) 52
ON TIMEOUT 53
OUTPUT 54
PLOTTER IS 55
PRINTER IS 56
PROTECT 57
PURGE 59
RENAME 60
RE-SAVE 61
RESET 62
RE-STORE 63
SAVE 64
SCRATCH A 65
STATUS 66
STORE 67
STORE SYSTEM 68
SYSTEM$ 69
TRANSFER 70
UNLOCK 71
SRM BASIC Error Codes for HP Series 200 Computers 72

Chapter 2: HP Series 200 Pascal Workstation Use on SRM 73
System Concepts 74
How Your Workstation Connects to SRM 74
Your Workstation's Identification 74
The Connection to the SRM Controller 74
The Controller's Identification 74
Identification of Shared Disc(s) 74
Unit Numbers for Shared Disc(s) 75
Identification of Shared Peripherals 75
SRM's Hierarchical Directory Structure 76
Notation 76
SRM Access Rights 77
SRM Concurrent File Access 78
Duplicating Files in More Than One Directory 79
Using Your Pascal Workstation on SRM 80
Moving Up and Down the Hierarchy 80
Using the Prefix Command 81
The Unit Directory Command 83
Moving Up the Hierarchical Directory Structure 85
Creating Directories 85
Specifying Files, Directories and Volumes 86
Syntax of File or Directory Specification 86
Syntax of a Volume Identifier 86
Passwords 87
Syntax of a Directory Path 87
SRM File or Directory Names 88
File Size Specification 88
Allowable File or Directory Names 88
Protecting Access to Files and Directories 89
Using Shared Printers and Plotters 90
Using Pascal Filer Commands With SRM 91

Chapter 3: System Startup 93
Initial System Startup From a BASIC SRM Workstation 94
Creating Directories on the System Disc 94
Planning the SYSTEMS Directory 96
If BASIC is the Predominant Operating System 97
If a Variety of Operating Systems are Used 98
Placing System Files in the SYSTEMS Directory 101
System Files Important for BASIC Use on SRM 101
Example: Placing the Loader Utility in the SYSTEMS Directory 101
Creating BASIC Workstation Bootup Configuration Files 102
Creating Configuration Files for Use by the Loader Utility 102
Creating Autostart Files for Use by the BASIC 3.0 System 103
Initial System Startup From a Pascal SRM Workstation 104
Before You Begin 104
Prerequisites 104
Boot ROM Versions 104
SRM Version 1.0 Operating System Parameters 105

Planning Your SRM Directory Structure 105
Overview of SRM Installation 107
Installing the SRM Driver Modules 107
Re-Configuring with TABLE 108
Creating Required Directories 108
Copying the System Files to SRM 109
Duplicating Links to System Files 110
SRM as the System Volume 111
Adding Modules to INITLIB 111
Replacing INITLIB 112
With Boot ROMs Version 3.0 and Later 112
With Earlier Version Boot ROMs 112
Multi-Disc SRM 113
CTABLE Modifications 114

Appendix A: Glossary 117

Appendix B: SRM Interface STATUS Registers 119

Appendix C: SRM and BASIC 2.0 121
System Startup From a BASIC 2.0 SRM Workstation 122
BASIC 2.0 Language Features 125
RE-STORE BIN 126
STORE BIN 127
Additional BASIC 2.0 Information 128
Modifying Existing Programs to Access Shared Printers or Plotters 128
Spooling 128
Formatted Output 129

Introduction

The Shared Resource Management (SRM) system allows several "workstation" computers to access shared mass storage and output devices (printers and plotters). This manual describes the use of HP Series 200 computers as SRM workstations.

Both the BASIC and Pascal language systems support use of SRM. The Pascal system incorporates features to access shared resources, while a BASIC BIN file (SRM) accommodates access to SRM from the BASIC language system.

About Revision Numbers

This manual contains references to the different versions of both the SRM operating system and of the BASIC and Pascal language systems supported on the HP Series 200 workstations. To clarify:

- The most current version of the SRM operating system is version 2.0. This manual describes the use of this latest version of the SRM system. Information for users of version 1.0 is noted in the text where required.
- SRM works with both BASIC 2.0 and BASIC 3.0. This manual describes the use of SRM with BASIC 3.0. The differences between uses of the two language system versions are summarized in the "SRM and BASIC 2.0" appendix.
- SRM works with both Pascal 2.1 and Pascal 3.0. This manual's descriptions of Pascal use with SRM applies to either version.

Additional References

If you are using your Pascal workstation for the first time, you should first read the *Pascal User's Guide* and the *Pascal Workstation System* manual (previously the *Pascal User's Manual*).

If you are not already familiar with the BASIC language system, you should refer to the *BASIC User's Guide* and other manuals shipped with that language system. This manual's "BASIC Language Reference" section supplements, but does not replace information in the *BASIC Language Reference* manual.

This Manual's Contents

This manual consists of three chapters and three appendices:

Chapter 1: HP Series 200 BASIC Workstation Use on SRM describes the use of HP Series 200 computers operating under the BASIC language system.

This chapter includes a conceptual overview of SRM, a tutorial demonstrating common uses of the workstation on SRM, and a language reference describing BASIC commands and statements that access shared resources.

Chapter 2: HP Series 200 Pascal Workstation Use on SRM outlines the use of HP Series 200 SRM workstations operating under the Pascal language system.

This chapter includes a conceptual overview of Pascal features that accommodate SRM use and a section demonstrating common procedures you'll need to use your SRM Pascal workstation.

Chapter 3: System Startup describes the procedures required to bring up HP Series 200 workstations on the SRM system for the first time. Presented for the SRM system manager, this chapter discusses recommended SRM directory structures for both BASIC and Pascal workstation use and describes installation of the BASIC or Pascal operating system software that allows workstations to access the SRM.

Appendix A is a **Glossary** of SRM terms.

Appendix B lists and describes **SRM Interface STATUS Registers**.

Appendix C: SRM and BASIC 2.0 summarizes the differences between the use of BASIC 2.0 with SRM and the use of BASIC 3.0 with SRM.

HP Series 200 BASIC Workstation Use on SRM

Chapter 1

This chapter describes the use of your HP Series 200 BASIC workstation with a Shared Resource Management system. The chapter is divided into four major sections:

- The **System Concepts** section is an overview to help you understand how the SRM system works.
- The **Using Your BASIC Workstation on SRM** section demonstrates, through the use of an example directory structure, some of the common operations involving shared resources.
- The **Modifying Existing Programs** section discusses ways to change existing BASIC programs to make them work with SRM.
- The **BASIC Language Reference for HP Series 200 SRM Workstations** section describes the use of BASIC commands and statements on SRM, including the special file and directory specification used with SRM.

System Concepts

This section presents a detailed look at some of the concepts of the SRM system, including descriptions of the following topics:

- support of the BASIC language on SRM;
- SRM directory structure and capabilities;
- storing of remote directories and files;
- shared access to directories and files (including file locking and password protection);
- management of shared peripherals.

Shared Resource Support of the BASIC Language

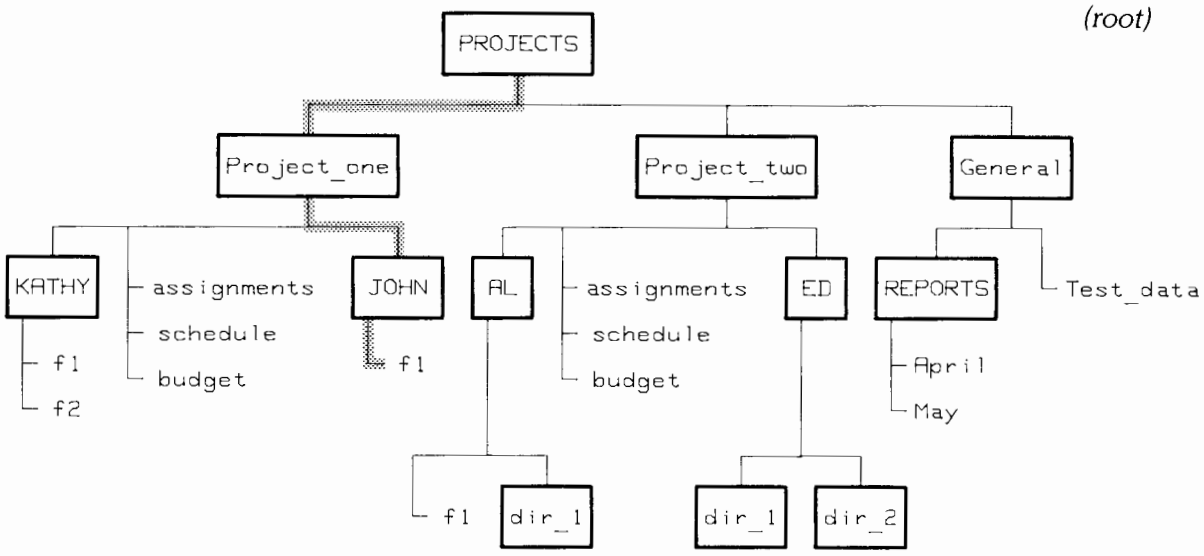
With HP Series 200 workstations, you can use most BASIC statements that access local mass storage devices to access shared mass storage devices on SRM as well. Any changes to BASIC mass storage statements made by the SRM BIN file are described in the "BASIC Language Reference" section of this chapter.

SRM adds three new commands to the BASIC mass storage statements used by HP Series 200 computers -- CREATE DIR, LOCK, and UNLOCK --and adds the PROTECT option for use with the CAT statement. In addition, the PROTECT statement's use on SRM is distinct from its use with local files.

SRM's Hierarchical Directory Structure

A directory is a file that is used to organize and control access to other files. The SRM operating system uses a hierarchical directory structure to organize and control access to files on a shared mass storage device.

As the word "hierarchy" suggests, directories are arranged in a series of "graded levels." Directories may contain either files or other directories. A file or directory within a directory is said to be "subordinate" to the containing directory. A directory is "superior" to the files and directories it contains.



In the illustration above, the directory named *KATHY* is subordinate to the directory named *Project_one*, because *Project_one* contains the information describing *KATHY*. The directory named *PROJECTS* is at level 1, the "root" level. You cannot create a directory at a higher level than the root level.

Each directory keeps information, in 24-byte fixed format records, about each file or directory immediately subordinate to it.

Uses of the Hierarchy: An Example

Suppose you're managing several projects, each of which needs to access a shared disc. To organize the files for each project separately, you can create a directory for each project (as shown in the illustration). Within each project directory, you can have a subordinate directory for each person working on the project as well as files to be shared among all users. Each person may then construct a directory/file system for organizing their own files.

Because files at different locations in the directory structure can have the same file name, you can use generic file names to identify similar project functions in the different projects. At the same time, the division into separate directories isolates the projects, and thus their individual functions, from one another. For example, Project_one's *budget* file is distinct from Project_two's *budget* file.

Directories also limit the number of files users must deal with at any one time. For example, people working on *Project_one* (see illustration) need never see the files in *Project_two* and may, in fact, confine most of their activity to within their own directories.

To maintain security, SRM provides the capability to protect access to directories and files. For example, you may wish to allow only members of a project team to read that project's files. Or, you may wish to prevent other users from altering the contents of a personal file.

In the first situation, you would protect the project directory's READ capability. By protecting a directory, you automatically restrict access to all directories and files subordinate to that directory. In the second situation, you would protect the file's WRITE capability. The section on "Shared Access to Remote Directories and Files" discusses protection in more detail.

Capabilities of Directories

Directories are a type of file and, as such, can be:

- created with the CREATE DIR statement. When a directory is created, its location in the hierarchical structure is fixed.
- cataloged with the CAT statement, renamed with the RENAME statement, and protected with the PROTECT statement.
- "filled" with subordinate files and directories using the COPY, CREATE BDAT, CREATE ASCII, CREATE DIR, SAVE, STORE, RENAME, RE-SAVE, and RE-STORE statements. Each subordinate file or directory is described in a 24-byte record in its superior directory.
- opened and closed with the MASS STORAGE IS (MSI) statement. When a user's MSI statement specifies a directory, any previously opened directory of that user is closed and the new one is opened.
- "emptied" by removing all subordinate files and directories with the PURGE statement.
- purged with the PURGE statement. You must close and empty a directory before purging it.

Referring to Directories and Files in the Hierarchy

To access either a directory or a file, you must specify its location in the hierarchical directory structure. This location is specified by a list of directories, called a directory path, that you must follow to reach the desired file or directory. Directory names in the list are delimited by a slash (/).

For example, in the directory structure illustrated previously, the remote file specifier:

```
"/PROJECTS/Project_one/JOHN/f1"
```

defines the "path" to the file, *f1*, through its superior directories.

The path to a file begins either at the root level or at the current working directory. The working directory is the directory specified by the most recent MASS STORAGE IS statement.

This chapter's "BASIC Language Reference" section discusses the rules for specifying remote files and directories.

How the SRM System Stores Remote Directories and Files

To most efficiently use the shared disc space, the SRM system stores files non-contiguously and adds to space allocations for files as needed.

Non-Contiguous Storage of Remote Files

To avoid wasting disc space, the SRM system may fragment a file to fill unused disc sectors. This process is transparent and cannot be externally controlled. By "filling the gaps" automatically, the system eliminates the need to pack the shared disc's files.

Space Allocation for Remote Directories and Files

SRM files and directories grow dynamically as data is entered into them.

Rather than restricting a file's space to that allocated when the file is created (for example, with a CREATE statement), the SRM system determines disc space requirements when data is sent to the file (for example, by an OUTPUT statement). If additional data placed into a file would cause the file to overflow its current space allocation, the system automatically allocates more space for the file.

Similarly, directories grow only as entries are added. As a file or directory is created, another 24-byte record is added to the containing directory.

Files are extended as long as there is sufficient unused disc space on the same volume. Excess data from a file will not be placed on any other disc (volume) on the SRM system.

Shared Access to Remote Directories And Files

Because the sharing of files is a consequence of shared mass storage, the SRM system provides features for controlling access to shared information.

Controlled Access: Password Protection

The SRM system offers three kinds of access capability for files and directories: READ, WRITE, and MANAGER. Capabilities are either public (available to all workstations on the SRM) or protected (available only to users who know the appropriate password).

Capabilities are protected with the PROTECT statement, which associates password(s) with one or more access capabilities. One password can be used to protect one or more capabilities. Each file or directory can have several password/capability pairs assigned to it.

Once assigned, the password protecting an access capability must be included with the file or directory specifier to execute statements requiring that access. If you don't specify the correct password when it is required, the system will report an error and deny access to the file or directory.

READ access capability for a file allows you to execute statements that read the file. READ access capability for a directory allows you to execute statements that read the file names in the directory, and to "pass through" the directory when the directory's name is included in a directory path.

For example, in the remote file specifier

```
"/PROJECTS/Project_one<READpass>/JOHN/f1"
```

including the assigned password <READPass> allows passage through the directory *Project_one* to allow access to its subordinate directories and files.

WRITE access capability for a file permits you to execute statements that write to the file. WRITE access capability for a directory allows you to execute statements that add to or delete from the directory's contents.

With the MANAGER access capability, public capabilities for a file or directory differ slightiy from password-protected capabilities. Public MANAGER capability allows any SRM user to PROTECT, PURGE or RENAME the file. The password-protected MANAGER capability provides MANAGER, READ and WRITE access capabilities to users who include a valid password in the file or directory specifier.

The "BASIC Language Reference" section in this chapter includes a table indicating the access capabilities needed to use each of the supported BASIC keywords. The description of the PROTECT keyword, also in that section, gives more details on protecting access to files and directories.

Exclusive Access: Locking Files

Although sharing files saves disc space, allowing several users access to one copy of a file introduces the danger of users trying to access the file at the same time, which can cause unpredictable results. For instance, if one user tries to read part of a file while another user is writing to it, the file's contents may be inaccurate for the read.

To avoid problems, the SRM system adds two BASIC keywords, LOCK and UNLOCK, which you can use to secure files during critical operations. LOCK establishes exclusive access to a file, which means that the file can only be accessed from the workstation at which the LOCK was executed. You may wish to LOCK a file, for example, during any procedure that writes new information to the file.

To permit shared access to the file once again, UNLOCK must be executed from the same workstation, or the file must be closed. Only ASCII or BDAT files that have been opened by a user via ASSIGN may be locked explicitly by that user.

Locking and unlocking is usually done from within a program. For more information, refer to the descriptions of the ASSIGN, LOCK and UNLOCK keywords in the "BASIC Language Reference" section of this chapter.

How the SRM System Manages Shared Peripheral Use

The SRM system not only provides shared access to printers and plotters, but also manages their use so that workstations never need to wait for output to be generated.

To use shared peripherals, you place files to be output into a special directory where they are held until the printer or plotter is free. The system keeps track of the order in which files arrive from the workstations, and outputs them in the same order. This method is called "spooling," and the directory where the files are kept is called the "spooler directory." Spooler directories are created for the SRM controller's use when the shared peripherals are installed on the SRM system.

After a file is placed in a spooler directory, the workstation is free to do other processing.

Using Your BASIC Workstation on SRM

This section describes, through examples, some of the more common procedures you'll use when operating your BASIC workstation on the SRM, including:

- booting from the SRM;
- accessing the shared mass storage device;
- creating directories and files;
- listing a directory's contents;
- copying files;
- using shared printers and plotters;
- protecting files and directories;
- purging files and directories;
- accessing files created on non-Series 200 SRM workstations;
- locking and unlocking files;
- returning to local mass storage.

This section illustrates both operations executed from the keyboard, and those executed within programs.

Note About Key References

Throughout this section, symbols for the keys used to execute statements and commands are shown with each statement or command.

The (EXECUTE) symbol denotes the execution key on either the HP 98602A or HP 98602B keyboards (the keycap on the HP 98602A keyboard is labeled (EXEC)). The (Return) symbol denotes the execution key on the HP 46020A keyboard.

You may also use the (ENTER) key on these keyboards to execute statements and commands.

Booting From the SRM

If your workstation has Boot ROM version 3.0 or later, you will be able to boot the BASIC language system into your workstation from the SRM. Once your workstation has been installed on the SRM system, the workstation powerup scheme your system manager has implemented on your SRM determines the exact procedure you use. This section discusses some general aspects of booting SRM workstations.

Note

Only HP Series 200 computers with Boot ROM version 3.0 or later can boot automatically from SRM. Refer to the *BASIC User's Guide* for more information on how to determine which boot ROM your computer has. Boot ROM 3.0L does **not** support automatic booting from SRM.

If your workstation's boot ROM does **not** support booting from SRM, you must boot the BASIC system from a local mass storage device and load the SRM and DCOMM BIN files to allow the workstation to communicate with the SRM system. You may load these BIN files either from local mass storage or, if your boot ROM supports automatic booting, from the SRM (even though the SRM BIN file is not present in the workstation).

For example, assume the SRM and DCOMM BIN files are in the directory named SYSTEMS at the root level of the SRM directory structure, and your workstation booted the BASIC system from the SRM. To load the BIN files from the SRM, you would type:

```
LOAD BIN "/SYSTEMS/SRM"
```

EXECUTE or Return

then type:

```
LOAD BIN "/SYSTEMS/DCOMM"
```

EXECUTE or Return

If you load the SRM and DCOMM BIN files from the SRM, you must load SRM before DCOMM.

Selecting an Operating System

In general, when you power your workstation ON or perform a SYSBOOT while the workstation is powered (which returns control to the boot ROM to restart the system selection and configuration process), you can either select the BASIC system explicitly or an operating system is loaded automatically.

If your workstation is not set up to automatically boot the BASIC system, you must explicitly select a system for the boot ROM to load into your workstation. Because explicit selection overrides any other method of system selection, you may choose this method over automatic selection when you wish to use an operating system other than the BASIC system.

To explicitly select an operating system for the boot ROM to load at powerup, follow these steps:

1. If your workstation's power is OFF, turn the power ON. To boot while the power is ON, use the SYSBOOT command (described in the *BASIC Language Reference*).

Note

If your workstation is providing power to an SRM multiplexer, you should avoid turning the power off to reboot.

---

2. Press any key within the first few seconds after the boot ROM's initial activity begins (the workstation's display begins to list the various parts of the computer for example, `Keyboard`) as each is recognized by the boot ROM). In response to the key press, the boot ROM then lists all systems currently available for loading into the workstation and waits for you to select a system.
3. To the left of each system name is a two-character identifier, such as `1B`. To select a system, type the identifier and wait for the boot ROM to load the specified system.

Automatic Configuration

Besides automatic selection of the boot system, your workstation may have an automatic configuration ("autostart") file, which specifies operations to be performed by the BASIC system immediately after it is loaded. For example, your workstation's autostart file may cause the system to load certain BIN files and go directly into your directory each time you boot your system.

If an autostart file exists for your workstation, all initial configuration happens automatically, without any extra effort from you. For information on setting up autostart files, refer to the *BASIC User's Guide* and the "Entering, Running and Storing Programs" chapter of the *BASIC Programming Techniques* manual.

Accessing the Shared Mass Storage Device

Your workstation accesses shared resources through the SRM controller, which is connected to the workstation through an HP 98629A interface in the workstation. The remote (SRM) mass storage device is identified by a remote mass storage unit specifier, or "remote msus" (similar to the local msus), which gives information about the SRM connection. The remote msus includes the following required and optional information:

- the device type REMOTE, which specifies the SRM system;

Note

Instead of the REMOTE device type specifier, you may use the "generic" form of the remote msus. Refer to the description of generic remote msus in the "BASIC Language Reference" section of this chapter.

- (Optional) the interface select code of your workstation's SRM interface. The default is the select code of the interface through which the boot ROM activates your workstation. (If you do not boot from the SRM, the default is the lowest select code of those available among the SRM interfaces in your workstation.)
- (Optional) the controller's node address;
- (Optional) the volume name and volume password.

The full syntax of the remote msus is described at the beginning of this chapter's "BASIC Language Reference" section.

In general, the first step in accessing a mass storage device is to make that device the MASS STORAGE IS device. Typing:

```
MSI ":REMOTE"
```

EXECUTE or Return

establishes the shared mass storage device as your workstation's mass storage and causes the root to be the working directory. The working directory is the directory specified in the most recent MSI statement. (Refer to the section on "System Concepts" earlier in this chapter for more information about directories.)

The form of the MSI statement shown above assumes that you want remote mass storage established according to the default values for your workstation's interface select code, the controller's node address, and the SRM system volume.

To find out the default values for these items, and to verify that your workstation's mass storage is the SRM mass storage device, you can use the CAT statement to list the contents of the working directory. Your mass storage is the remote device if, when you type:

```
CAT
```
EXECUTE or Return

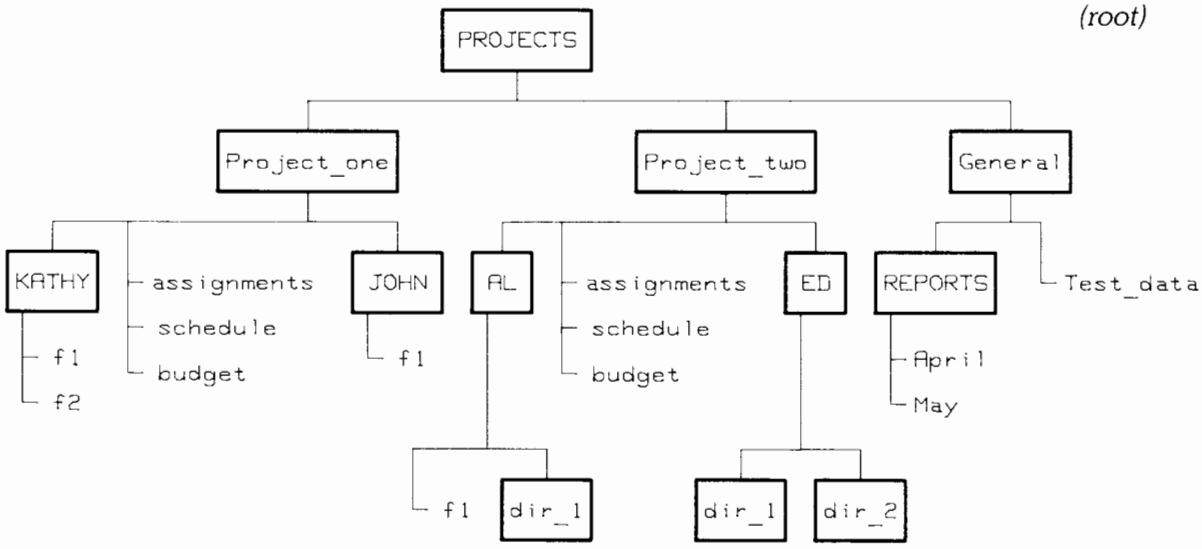
the directory header includes the remote msus (for example, `:REMOTE 21, 0`). Refer to the CAT keyword entry in this chapter's "BASIC Language Reference" section for an example of a remote directory catalog listing. If, as in this example, you do not specify the optional items in your remote msus, the default values are assumed and listed.

To specify the remote mass storage when the SRM controller's node address is 4 and the select code of your workstation's interface is 15, you would type:

```
MSI ":REMOTE 15,4"
```
EXECUTE or Return

Creating Directories and Files

For the following examples, assume you are working with the directory structure shown in the illustration below.



Creating Directories

To create a directory named *CHARLIE* in the directory, *Project_one*, you could type:

```
MSI ":REMOTE"
```
EXECUTE or Return
```
CREATE DIR "/PROJECTS/Project_one/CHARLIE"
```
EXECUTE or Return

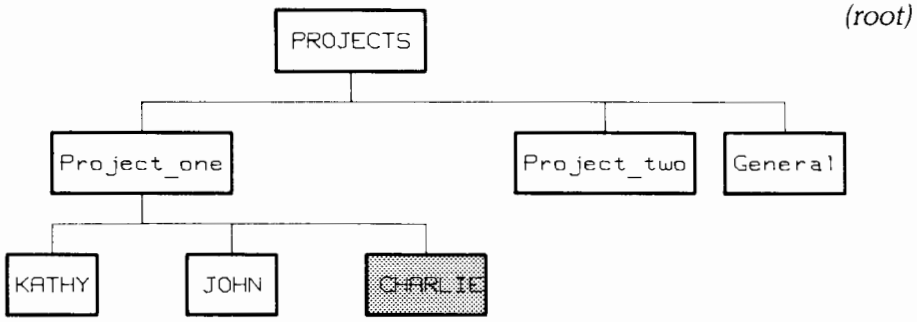
The leading slash indicates that the directory path begins at the root of the SRM directory structure.

You could accomplish the same thing by typing:

```
CREATE DIR "PROJECTS/Project_one/CHARLIE:REMOTE"
```
EXECUTE or Return

Using the leading slash to begin the directory path at the root works only if you have previously established the remote mass storage as your workstation's mass storage (with some form of the `MSI ":REMOTE"` statement).

This statement would place your newly-created directory into the directory structure as shown below.



Creating Files and Other Directories Under a Directory

To create files subordinate to a new directory, you may either establish the new directory as the working directory or specify the directory path to that directory. Assuming your current working directory is the root, you could type:

```
MSI "PROJECTS/Project_one/CHARLIE"
```
EXECUTE or Return

to move into the directory, *CHARLIE*.

You could verify the new working directory with a catalog listing by typing:

```
CAT
```
EXECUTE or Return

On a computer whose screen supports an 80-character line width, the resulting listing would look something like this:

```
PROJECTS/Project_one/CHARLIE:REMOTE 21, 0
LABEL:     Disc1
FORMAT:    SDF
AVAILABLE SPACE:        54096
                          SYS  FILE   NUMBER   RECORD      MODIFIED      PUB OPEN
FILE NAME            LEV TYPE  TYPE  RECORDS   LENGTH DATE         TIME ACC STAT
==================== === ==== ===== ======== ======== ================ === ====
```

To create an ASCII file within *CHARLIE*, which is named *ASCII_1* and is initially to contain 100 records, you would type:

```
CREATE ASCII "ASCII_1",100
```
EXECUTE or Return

To create a BDAT file within *CHARLIE*, which is named *BDAT_1* and is initially to contain 25 records, you would type:

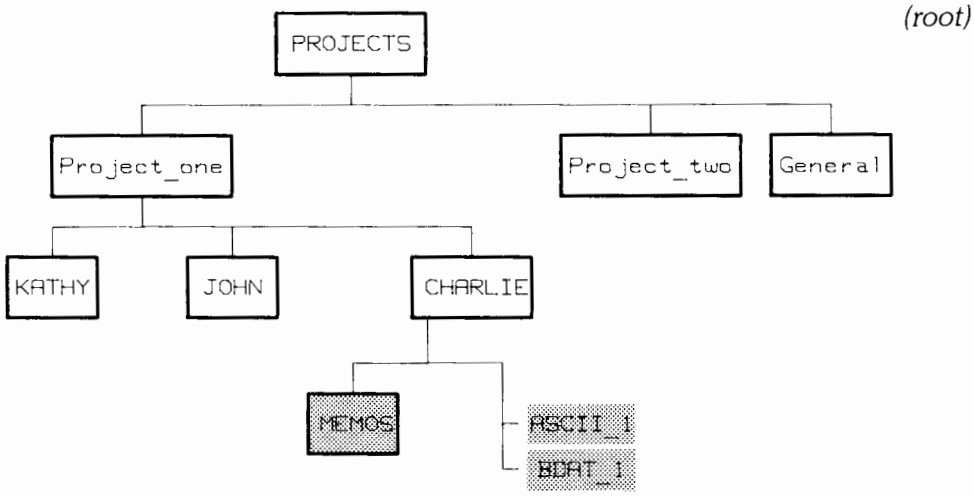
```
CREATE BDAT "BDAT_1",25
```
EXECUTE or Return

(When no record size is specified in the CREATE BDAT statement, the default 256-byte record size is assumed.)

To create another directory within *CHARLIE* called *MEMOS*, you would type:

```
CREATE DIR "MEMOS"
```
EXECUTE or Return

The additions would make the directory structure look like this:



The simplest form of the CAT statement:

```
CAT
```
EXECUTE or Return

lists the contents of the current working directory because no directory is specifically identified. If no directory name is shown in the directory header, the current working directory is the root.

If you wanted to list the contents of *CHARLIE*, but your current working directory was **not** *CHARLIE*, you could:

- Designate *CHARLIE* as the working directory with the MSI statement, then use the CAT statement's "short form." For example:

  ```
  MSI "PROJECTS/Project_one/CHARLIE:REMOTE"
  ```
  EXECUTE or Return
  ```
  CAT
  ```
  EXECUTE or Return

- In the CAT statement, specify the entire path to *CHARLIE*, starting at the root, by beginning the path name with a slash (/). For example:

  ```
  CAT "/PROJECTS/Project_one/CHARLIE"
  ```
  EXECUTE or Return

This form assumes that you have already designated remote mass storage with some form of the MSI ":REMOTE" statement. If you have not, use the form:

```
CAT "PROJECTS/Project_one/CHARLIE:REMOTE"
```
EXECUTE or Return

The leading slash is not necessary, because including :REMOTE specifies the root as the beginning of the path.

- If you were in *MEMOS* (the directory immediately subordinate to *CHARLIE*), you could use the " .. " notation (explained with directory path syntax in the "BASIC Language Reference" section of this chapter). For example:

```
CAT ".."
```
EXECUTE or Return

For more details on specifying remote files and directories in BASIC statements, refer to the beginning of the "BASIC Language Reference" section in this chapter.

Copying Files

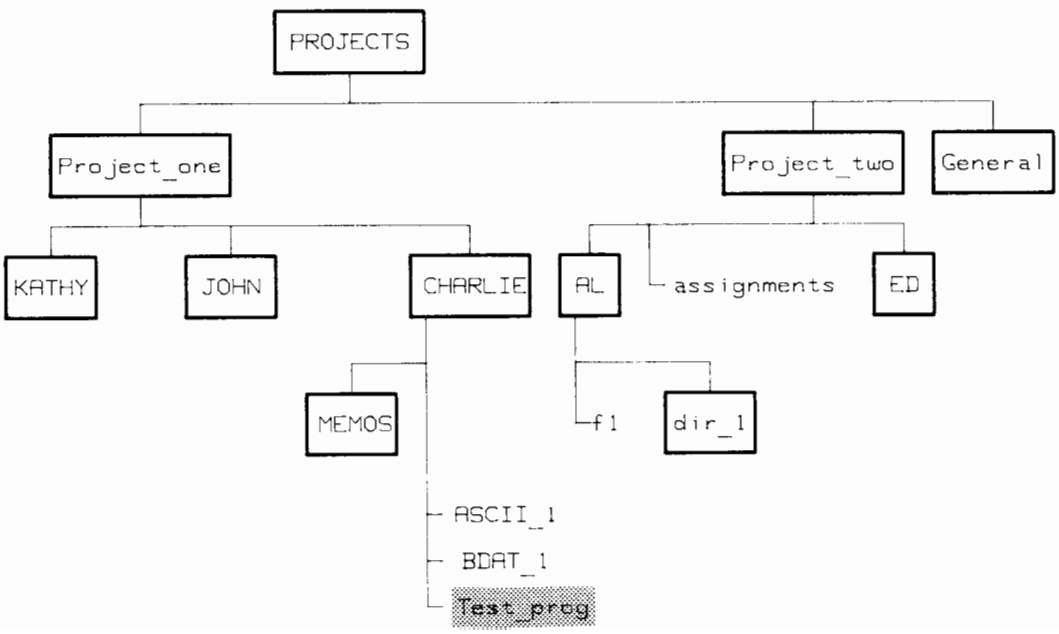
With SRM, you can copy files between local and remote mass storage devices by any of the methods illustrated in the following examples. Again using the directory structure established for the other examples in this section, assume that the current working directory is *CHARLIE*.

Using the COPY Statement

The most direct method of copying a file from local to remote mass storage is to use the COPY statement. For example, to copy a PROG file named *Test_prog* that is on a local disc drive into the directory *CHARLIE* on the SRM system disc, you could type:

```
COPY "Test_prog:INTERNAL" TO "Test_prog"
```
EXECUTE or Return

By including the :INTERNAL msus, you can access the local mass storage without changing the current working directory (which is a remote directory). Refer to the "Data Storage and Retrieval" chapter of the *BASIC Programming Techniques* manual for information on alternatives to the :INTERNAL msus for specifying local mass storage.



Other Uses of COPY

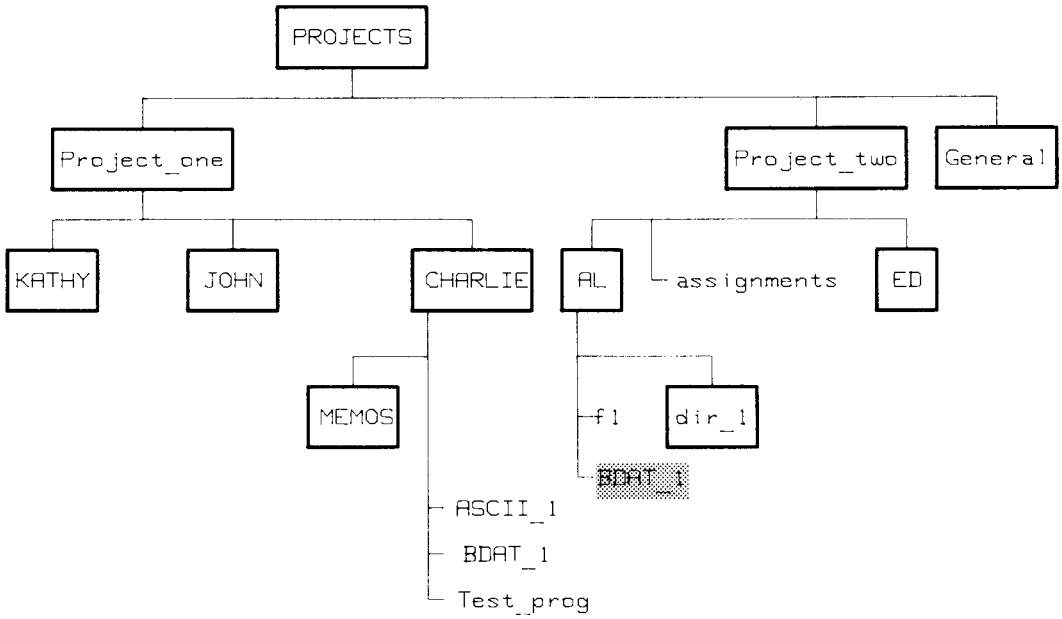
The COPY statement can be used to copy files not only from local to remote mass storage but also from remote to local mass storage and from one remote mass storage device to another. You cannot copy directories, although you can copy files from one directory to another. Similarly, you cannot copy an entire remote mass storage volume in a single COPY statement. (You must copy a remote volume file by file.)

Suppose you want to copy the file *BDAT_1* from the directory *CHARLIE* into the directory *AL* (see previous illustration).

Assuming the working directory is *CHARLIE*, you could type:

```
COPY "BDAT_1" TO "/PROJECTS/Project_two/AL/BDAT_1"
```
EXECUTE or Return

The effect of the copy on the directory structure is illustrated below:



Using LOAD and STORE

You may also copy files by loading the program into your workstation from local mass storage and then storing it in remote mass storage. For example, to copy a PROG file named *Test_prog* that is on a disc in your workstation's built-in disc drive into the directory *CHARLIE* on the SRM system disc (as demonstrated earlier using COPY), you could type:

```
LOAD "Test_prog:INTERNAL"
```
EXECUTE or Return

Once the file is in your workstation's memory, you may then store the file in the remote directory by using a statement such as:

```
STORE "Test_prog"
```
EXECUTE or Return

Copying Item-by-Item Using ENTER and OUTPUT

You may also copy a file from local to remote mass storage an item at a time, as illustrated in the programs that follow. These programs use the ENTER and OUTPUT statements to copy data item-by-item from a local BDAT file to remote mass storage.

The first program creates and fills a BDAT file named *BDAT_FILE*.

```
10      CREATE BDAT "BDAT_FILE:INTERNAL",10
20      ASSIGN @Local TO "BDAT_FILE:INTERNAL"
30      !
40      FOR Item=1 TO 50
50      OUTPUT @Local;"String data item"
60      NEXT Item
70      !
80      ASSIGN @Local TO *
90      END
```

The second program copies the contents of *BDAT_FILE* item-by-item into a file (also called *BDAT_FILE*) in the SRM directory named *General* (shown in the previous illustration).

```
100     DIM String_item$[20]
110     CREATE BDAT "PROJECTS/General/BDAT_FILE:REMOTE",10
120     ASSIGN @Local TO "BDAT_FILE:INTERNAL"
130     ASSIGN @Remote TO "PROJECTS/General/BDAT_FILE:REMOTE"
140     !
150     FOR Item=1 TO 50
160     ENTER @ Local;String_item$
170     OUTPUT @Remote;String_item$
180     NEXT Item
190     !
200     ASSIGN @Local TO *
210     ASSIGN @Remote TO *
220     END
```

Using a Shared Printer or Plotter

Use of special SRM directories called "spooler directories" allows you to access a shared printer or plotter. Setting up a spooler directory is explained in the "Interfaces and Peripherals" chapter of the *SRM Operating System Manual*. The examples in this section assume that the spooler directories *LP* (for "Line Printer") and *PL* (for "PLotter") have been created at the root of the SRM directory structure.

Spooling Using PRINTER IS and PLOTTER IS

You can use the PRINTER IS and PLOTTER IS statements to send data to your shared printer or plotter. The following command sequence illustrates this spooling method:

```
CREATE BDAT "/LP/Print_file",1
PRINTER IS "/LP/Print_file"
LIST
XREF
PRINTER IS CRT
```

PRINTER IS and PLOTTER IS work only with BDAT files. Because the SRM 1.0 operating system's spooling works only with ASCII files, you cannot use PRINTER IS and PLOTTER IS for spooling with that version of SRM.

Note

The DUMP DEVICE IS and PRINTALL IS statements do **not** support files, so cannot be used for printer spooling.

Writing Files to the Spooler Directories

You may also access the printer associated with *LP* by placing the data to be printed in an ASCII or BDAT file in that spooler directory. For example, to list a program currently in memory, you could SAVE the program in *LP* as the file *P1_LISTING* by typing either:

```
SAVE "LP/P1_LISTING:REMOTE"
```
EXECUTE or Return

or

```
SAVE "/LP/P1_LISTING"
```
EXECUTE or Return

The SAVE statement creates an ASCII file. Although this is the same syntax used to save programs on a shared disc, the SRM system knows that *LP* is a spooler directory and prints the file as soon as possible.

Note

When used for spooling, SAVE places a file in the spooler directory. The file is printed, then purged. You may wish to save or create the file first, then use the COPY statement to place the file into the spooler directory.

Sending Program Output to a Shared Printer

To spool program output to a shared printer, create an ASCII or BDAT file, assign an I/O path name to the file (which opens the file), and OUTPUT the data to that file. With BDAT files, you should ASSIGN with FORMAT ON. When the file's contents are to be printed, close the file. The following example program segment outputs the data stored in the string array called *Data$* to an ASCII file named *PERFORMANCE*.

```
760   CREATE ASCII "/LP/PERFORMANCE",100
770   ASSIGN @Spool TO "/LP/PERFORMANCE"
780   OUTPUT @Spool;"Performance Summary"
790   OUTPUT @Spool;Data$(*)
800   ASSIGN @Spool TO *   ! Initiate printing.
```

The system waits until the file is non-empty and closed before sending its contents to the output device. If your file is not printed or plotted within a reasonable amount of time, you may not have closed it. You can verify that your file is ready to be printed or plotted by cataloging the spooler directory:

```
CAT "/LP"
```
EXECUTE or Return

The open status (`OPEN STAT`) of the file currently being printed or plotted is listed as locked (`LOCK`). Files currently being written to the spooler directory (either printer or plotter) are listed as `OPEN`. Files that do not have a status word in the catalog are ready for printing or plotting.

The SRM 2.0 operating system allows BDAT files to be sent to the printing device as a byte stream. (With SRM 1.0, only ASCII files can be used.)

Note

With the SRM 2.0 operating system, a BDAT file sent to the spooler is printed exactly as the byte stream sent. Unless you set up the BDAT file correctly, improper printer output or operation could result. Therefore, you should ASSIGN BDAT files with FORMAT ON before outputting data.

The spooler prints each string and numeric item on a separate line by inserting a carriage return and line feed after each item. To put several strings on one line, concatenate them into one string before using OUTPUT to send them to the spooler file. You may insert ASCII control characters in the data by using the CHR$ string function.

Appearance of Output

Printed output for each file includes a one-page header, which identifies the directory path to the file, the file's name, and the date and time of the printing.

To cause the printer to skip the paper perforation after printing a page (60 lines), prefix your file name with "FF". For example:

```
SAVE "/LP/FF_MYTEXT"
```
EXECUTE or Return

Preparing Plotters

If your plotter does not feed its paper automatically, a message appears on the SRM controller screen indicating that the plotter needs to be set up. After you have put paper on the plotter, you may begin the plotting by using the controller's SPOOL CONTINUE command (described in the *SRM Operating System Manual*). Plotters with automatic paper feed require no operator intervention.

Aborting Printing/Plotting In Progress

To abort an in-progress printing or plotting, use the SPOOL ABORT command from the SRM controller. The system stops sending data to the output device and closes, then purges the file. For details on bringing the spooler UP and DOWN, see the description of the SPOOLER command in the "Language Reference" section of the *SRM Operating System Manual.*

With the SRM 2.0 operating system, if a printer is taken off-line while a file is being printed, the printer stops and resumes when the printer is put back on-line. No data is lost during such an interruption. The SRM 1.0 operating system also resumes printing, but from the beginning of the file.

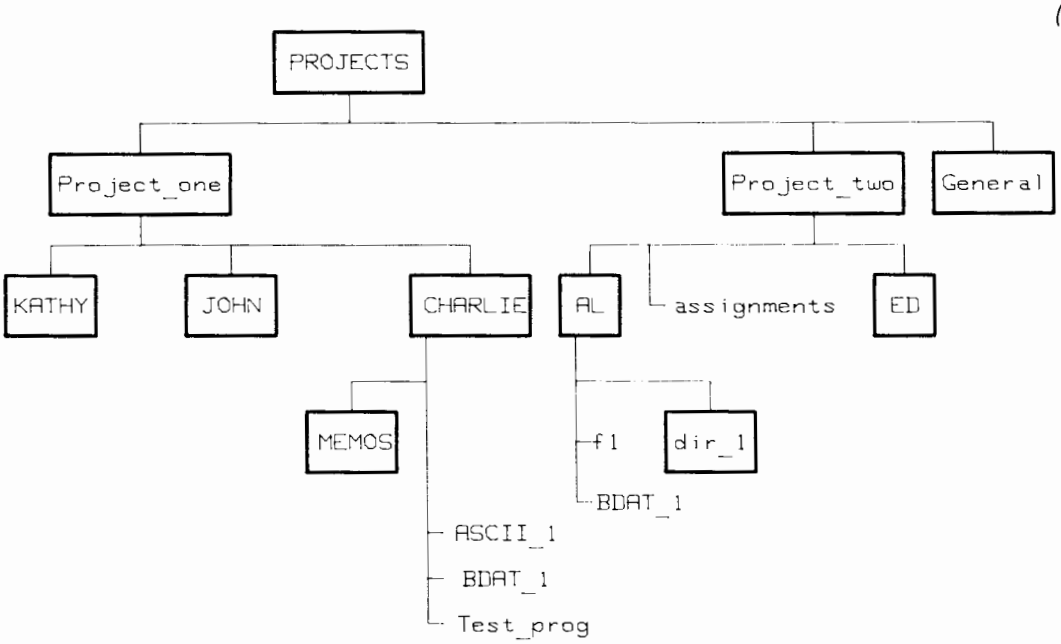
Protecting Files and Directories

When you create directories and files, their access capabilities are "public" (available to any user on the SRM). You may subsequently protect a directory or file against certain types of access by other SRM workstations, provided:

- you have MANAGER access capability on the file or directory (MANAGER access to the file is public or you know the password protecting the capability);

- you have READ access capability on the directory immediately superior to the file or directory you wish to protect;
- you protect the file or directory either while "in" its superior directory or by specifying the valid directory path to its superior directory.

For example, using the directory structure established for other examples in this section (see illustration) and assuming no passwords have been assigned to the files, you could:



1. Assign the password *passme* to protect the MANAGER and WRITE access capabilities on the directory *CHARLIE* with the sequence:

```
MSI "/PROJECTS/Project_one"        EXECUTE or Return
PROTECT "CHARLIE",("passme":MANAGER,WRITE)    EXECUTE or Return
```

   which executes the PROTECT statement after moving to the directory *Project_one* (immediately superior to *CHARLIE*). As a result of this PROTECT statement, the READ access capability on *CHARLIE* is still public, but any operations that require MANAGER or WRITE capabilities must include the password.

2. Remove all public access capabilities from the file *ASCII_1* by assigning the password *no_pub*, using:

```
PROTECT "CHARLIE/ASCII_1",("no_pub":MANAGER,WRITE,READ)    EXECUTE or Return
```

   or

```
MSI "CHARLIE"    EXECUTE or Return
PROTECT "ASCII_1",("no_pub":MANAGER,WRITE,READ)    EXECUTE or Return
```

These statements assume you are in the directory, *Project_one*, as if you had executed the statements in the previous step.

The second sequence of statements makes *CHARLIE* the new working directory, whereas in the first, you merely "pass through" *CHARLIE* to reach *ASCII_1*. With the READ access capability on *CHARLIE* still public, you do not need a password.

3. Protect the file, *BDAT_1*, so that data can be read from it but not written into it without using the password, *write*. If the current working directory were *CHARLIE*, you would type:

```
PROTECT "BDAT_1",("write":MANAGER,WRITE)
```
EXECUTE or Return

4. Protect the MANAGER access capability of the directory *MEMOS* with the password, *mgr_pass* (so that everyone can read from and write to the directory, but a password is required to purge the directory or its contents) by typing:

```
PROTECT "MEMOS",("mgr_pass":MANAGER)
```
EXECUTE or Return

If you protected the files and directory in *CHARLIE* as in the steps above, a catalog listing of *CHARLIE* would look something like this:

```
PROJECTS/Project_one/CHARLIE:REMOTE 21, 0
LABEL:     Disc1
FORMAT:    SDF
AVAILABLE SPACE:        54096

                           SYS  FILE    NUMBER    RECORD      MODIFIED      PUB OPEN
FILE NAME             LEV TYPE  TYPE   RECORDS    LENGTH DATE         TIME ACC STAT
==================== === ==== ===== ======== ======== ================ === ====
ASCII_1                 1       ASCII         0      256  2-Dec-84  13:20
BDAT_1                  1 98X6  BDAT          0      256  2-Dec-84  13:20  R
MEMOS                   1       DIR           0       24  2-Dec-84  13:20  RW
```

The letters in the column labeled PUB ACC indicate access capabilities that are public (not protected with a password). For example, only the MANAGER (M) access capability on the directory *MEMOS* has been protected, leaving the READ (R) and WRITE (W) capabilities available to any SRM workstation user.

Specifying Passwords

When a password is required, you must include the correct password as part of the file or directory specifier in any command or statement that requires the protected access on the file or directory. The password must be enclosed between " < " and " > " and must immediately follow the name of the file or directory it protects.

For example, to get the file *ASCII_1*, you might type:

```
GET "/PROJECTS/Project_one/CHARLIE/ASCII_1<no_pub>"
```
EXECUTE or Return

If the password were not included in the specifier, the system would respond with an error message and refuse to get the file.

Purging Remote Files and Directories

The PURGE statement works the same for removing remote files as for removing files from local mass storage. You may also remove directories using PURGE. PURGE works only with closed files and directories. Directories must also be empty (not contain any files or directories). Refer to the discussion on "Returning to Local Mass Storage" later in this section for details on closing files and directories.

When specifying the remote file to be purged, you must include all passwords protecting access capabilities required for the PURGE. For example, to purge the file *BDAT_1* from the directory *CHARLIE* (see previous examples), you could type:

```
PURGE ":<passme>/BDAT_1<write>"
```
EXECUTE or Return

In this example, *CHARLIE* is the current working directory, as denoted in the directory path by " : ". (Refer to the syntax for directory path in this chapter's "BASIC Language Reference" section.)

To purge a file, you must have the MANAGER access capability on that file and READ and WRITE access capabilities on the file's superior directory. Because *passme* protects the WRITE capability on *CHARLIE* and *write* protects the MANAGER capability on *BDAT_1*, both passwords must be included in the file specifier in the PURGE statement.

Although you do not normally need to specify the working directory in a directory path, you must include the password for the PURGE operation. The READ capability on *CHARLIE* is not password-protected.

To purge *CHARLIE*, you would first need to purge the remaining files and directory in *CHARLIE*. Because the MSI statement "opens" a directory (making it the current working directory), you must also "close" *CHARLIE*.

For example, if no files or directories remained in *CHARLIE*, you could purge *CHARLIE* by typing:

```
MSI ":REMOTE"
```
EXECUTE or Return
```
PURGE "PROJECTS/Project_one/CHARLIE<passme>"
```
EXECUTE or Return

The first statement closes *CHARLIE* and establishes the root directory as the current working directory. Note that, because *passme* protects the MANAGER access capability on *CHARLIE*, you must include that password in the PURGE statement.

Accessing Files Created on Non-Series 200 SRM Workstations

Regardless of the kind of the computer or language system, files containing ASCII data can be shared among all workstations on the SRM.

This example shows how you can access a remote ASCII file named *Prog_x*, which was created with the SAVE ASCII statement on an HP 9845 with the SAVE ASCII statement.

In this example, *Prog_x* is in an HP 9845 workstation user's directory called *COMMON*. *COMMON* is located in the directory *WORK_45*, which is at the root of the SRM directory structure. The password *mypass* protects the READ capability on *WORK_45*. All access capabilities on *COMMON* are public.

To access *Prog_x*, you would type:

```
GET "WORK_45<mypass>/COMMON/Prog_x:REMOTE"
```
(EXECUTE) or (Return)

or

```
GET "/WORK_45<mypass>/COMMON/Prog_x"
```
(EXECUTE) or (Return)

The system would then put *Prog_x* into your workstation. Keep in mind that, with GET, any lines containing BASIC syntax that is invalid in BASIC for the Series 200 computers will be stored as commented program lines (!).

Locking and Unlocking Remote Files

You can "lock" a shared file with the LOCK statement, giving you sole access to that file. The same file can be locked several times in succession. Unlocking a file requires that you cancel all locks on that file. If you use the UNLOCK statement, you must cancel each LOCK with a corresponding UNLOCK. Using ASSIGN to re-open a locked file unlocks the file and you must execute another LOCK statement to lock the file again. Closing the file via ASSIGN @...TO * cancels all locks on the file.

In this example, a critical operation must be performed on the file named *File_a*, and you do not want other users accessing the file during that operation. The program might be as follows:

```
1000   ASSIGN @File TO "File_a:REMOTE"
1010   LOCK @File;CONDITIONAL Result_code
1020   IF Result_code THEN GOTO 1010  ! Try again
1030   !  Begin critical process
       .
       .
       .
2000   !  End critical process
2010   UNLOCK @File
```

The numeric variable called *Result_code* is used to determine the result of the LOCK operation. If the LOCK operation is successful, the variable contains 0. If the LOCK is not successful, the variable contains the numeric error code generated by attempting to lock the file.

Returning to Local Mass Storage

When you have finished accessing shared resources, you should close all of your files and directories to "release" the system resources.

Remote files are closed by ASSIGN ... TO * (see this chapter's "BASIC Language Reference" section for details on ASSIGN). The SCRATCH A command closes directories and files. All remote files except those opened with the PRINTER IS statement are also closed by pressing (RESET).

To close your current working directory, MSI to a local msus (for example, `MSI ":INTERNAL"`).

If you booted from local mass storage, you may also execute the SCRATCH A command to completely release your access to the system. If you booted from the SRM, executing SCRATCH A resets the current working directory to the root.

Modifying Existing Programs to Access Shared Resources

This section summarizes ways you can modify existing programs that access local resources to allow those programs to access shared resources.

When modifying programs to access SRM-controlled mass storage device(s), you should be aware that:

- Local and remote mass storage file specifiers may differ and string variable names that contain file specifiers may need corresponding modification.
- References to mass storage unit specifiers (msus) throughout the program may have to be altered.
- Allowances may have to be made for directory path specification.
- Local protect codes may differ from passwords on remote files. The syntax for protecting remote files is different from that used for local files.

File Specifiers

Composition of File Names

All file names for local mass storage are one to 10 characters long, while remote file names contain one to 16 characters. Remote file names can contain the period character (.) while local files cannot. If file name compatibility between resources is required, use 10 or fewer characters and do not use periods within remote file names.

File and Mass Storage Device Specification in String Variables

Modifying programs for use with shared resources generally requires changing the value, and often the length, of the string variables used to specify files and mass storage devices. The statements that assign the actual values to the string variables may have to be modified individually.

Some programs use one string variable for the entire file specifier. For instance:

```
100   DIM File_specifier$[32]
110   LINPUT "Enter file specifier", File_specifier$
120   ON ERROR GOTO 110 ! Try again if improper specifier.
130   ASSIGN @Path TO File_specifier$
140   OFF ERROR
```

If one variable is used for all file specifiers (as in the preceding example), only the length of the variable needs to be changed to allow for the additional characters allowed in remote file specifiers.

The maximum number of characters that can be entered into a string variable from the keyboard in one operation is a good size for a file specifier variable. The Models 216, 220, 226 and 236 allow up to 160 characters and the Model 237 allows 256 characters. Thus, the length of `File_specifier$` in the preceding example's DIM statement would be changed from 32 to 160 or 256, accordingly.

Note that the system mass storage (the current MASS STORAGE IS device) will be accessed if no msus is explicitly specified.

Mass Storage Unit Specification

Some programs use separate variables for the file name and mass storage unit specifier. For example:

```
ASSIGN @Path TO Filename$&Msus$
```

If so, both variables may have to be dimensioned to greater lengths. Allowing 34 characters for the file name variable accommodates a 16-character file name, a 16-character password, and the "<" and ">" password delimiters (for example, "ASCDEFGHIJ123456<1234567890123456>"). The remote msus may occupy up to 54 characters.

Other programs may use MASS STORAGE IS statements throughout the program instead of including the msus in each file specifier. For instance:

```
MASS STORAGE IS Left_drive$
ASSIGN @File TO File_name$
```

Unless variable(s) are used to specify the msus and each variable is assigned a value in only one place, you may have to modify each MASS STORAGE IS statement to specify the desired remote mass storage device.

Allowing for Directory Paths

Suppose the following program needs to be modified to include a remote file's directory path.

```
100  DIM Filename$[14],Msus$[20]
      .
      .
      .
500  Filename$="SLIDES"
510  Msus$=":HP9895,700"
      .
      .
      .
1000 ASSIGN @File TO Filename$&Msus$
1010 OUTPUT @File;Data(*)
1020 ASSIGN @File TO *
      .
      .
      .
2000 ASSIGN @File TO Filename$&Msus$
2010 OUTPUT @File;Data(*)
2020 ASSIGN @File TO *
```

In this example, it is probably easiest to add another string variable for the (optional) directory path name. For example:

```
100  DIM Dir_path$[160],Filename$[80],Msus$[80]
     .
     .
     .
500  Dir_path$="FRED/DATA_FILES/"
510  Filename$="SLIDES"
520  Msus$=":REMOTE 21,1"
     .
     .
     .
1000 ASSIGN @File TO Dir_path$&Filename$&Msus$
1010 OUTPUT @File;Data(*)
1020 ASSIGN @File TO *
```

If the `Dir-path$` variable is null, the statement looks exactly like it did before the modification. If the `Msus$` variable is null, the current mass storage device is accessed. The only difference is in the allowable length of the string variables.

Passwords and Protect Codes

The PROTECT statement format for remote files is different form the format for local files. Depending on the type of mass storage is being used, you can use either of the following to decide which syntax will be used:

1. Try the non-SRM syntax with an ON ERROR statement enabled. If an error occurs, see if it indicates that the mass storage device is an SRM. An Error 1 occurs when the following statement is executed on a remote file.

   `PROTECT` *file specifier*`,`*protect code*

2. If the program uses a string to store the mass storage unit specifier, check for a non-zero value of `POS(Msus$,"REMOTE")`. This alternative is easier to implement than alternative 1 but will not work if the program accesses the default device when `Msus$` is empty.

If the program looks for a password error (Error 62) at ASSIGN time, the program may have to be modified because the system may not detect the password error until an ENTER @Path or OUTPUT @Path is attempted.

BASIC Language Reference for HP Series 200 SRM Workstations

This section lists all BASIC keywords either used exclusively with SRM or whose use with SRM differs from that described in the *BASIC Language Reference* manual.

Most keyword entries in this section describe only differences between the keyword's normal use and its use on SRM. Because SRM-specific keywords are not described in the *BASIC Language Reference* manual, full details of their use are included in this section.

You may wish to remove this language reference section and insert it as an appendix to your *BASIC Language Reference* manual. SRM-specific keywords (CREATE DIR, LOCK and UNLOCK) are described on separate pages, so you can insert those keywords into their correct alphabetical position in the main *BASIC Language Reference* manual. Be sure not to discard these pages if you replace your *BASIC Language Reference* manual with a newer version.

The primary difference in keyword syntax for SRM use is in file specification. Use of supported keywords on SRM requires you to supply a **remote file specifier** rather than the file specifier described for non-SRM uses of BASIC. Some keywords also involve a **directory specifier**, which is unique to SRM. Remote file specifiers and directory specifiers are described at the beginning of this section.

In addition, you must be aware of the **access capabilities** required on files and directories involved in the keyword's use. Access capability requirements are summarized in a table included in this section.

This section is not a complete language reference for the BASIC keywords listed. Instead, each keyword entry shows the SRM-specific use of the keyword and supplies details relevant to that use. The *BASIC Language Reference* manual completely describes the BASIC keywords.

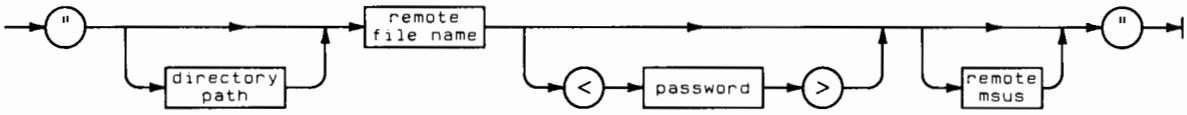
Note

This section reflects the keywords and uses for BASIC 3.0. The "SRM and BASIC 2.0" appendix lists differences between the BASIC 2.0 and BASIC 3.0 uses.

Syntax for Remote File and Directory Specification

The following syntax applies to remote file specification for BASIC keyword use on SRM. The semantics discussion applies to all remote file specification unless otherwise noted with a specific keyword's description.

Remote File Specifier



Item	Description/Default	Range Restrictions
directory path	literal	(see diagram)
remote file name	literal	any valid remote file name (see Semantics)
password	literal, first 16 non-blank characters are significant	any valid password (see Semantics)
remote msus	literal	(see diagram)

Semantics

A valid **remote file name** consists of one to 16 characters, which may include uppercase and lowercase letters, digits 0 through 9, the underbar (_) character, the period (.) character, and national language characters (CHR$(161) through CHR$(254)). Spaces are ignored.

A valid **password** consists of one to 16 characters, which may include any ASCII character except " > ". Spaces are ignored. Passwords are assigned by the PROTECT keyword.

If no directory path is included, the system assumes the file is in the current working directory (the directory specified in the latest MASS STORAGE IS statement). To specify a file in a directory other than the current working directory, specify the directory path to the desired file. (Refer to the syntax for directory path later in this section.) The directory path may begin at the current working directory or at the root.

The READ access capability for each directory included in the directory path must be public or the password that currently protects the READ capability must be included in the remote file specifier. A maximum of six identifiers can be included in a specifier -- five directories in the path and the target file. If the target file is more than five directories away from the current working directory, move closer by changing the working directory (with MSI).

Examples

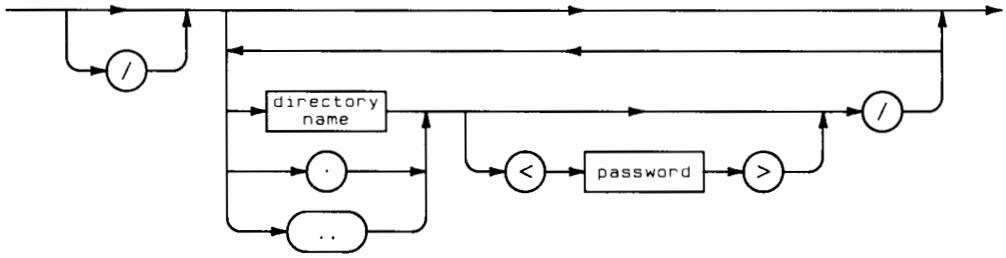
```
"PROJECTS/WRITERS/samples<wr_pass>:REMOTE 21,1;LABELVOL_TWO<master>"
```

illustrates the full remote file specifier syntax. For explanations of the directory path and remote msus portions of this illustration, see the examples with those components.

```
"thisfile"
```

specifies a file that is in the current working directory. This form assumes that the SRM (remote mass storage) has previously been "entered" via some form of the `MSI ":REMOTE"` statement.

Directory Path



Item	Description/Default	Range Restrictions
directory name	literal	any valid directory name (see Semantics)
password	literal, first 16 non-blank characters are significant	any valid password (see Semantics)

Semantics

A valid **directory name** consists of one to 16 characters, which may include uppercase and lowercase letters, digits 0 through 9, the underbar (_) character, the period (.) character, and national language characters (CHR$(161) through CHR$(254)). Spaces are ignored.

A valid **password** consists of one to 16 characters, which may include any ASCII character except " > ". Spaces are ignored. Passwords are assigned by the PROTECT.

A leading slash (/) in the directory path specifies that the path begins at the root. If you have not previously established the remote mass storage (using, for example, `MSI ":REMOTE"`), you must include some form of the remote msus with the file specifier. Including the remote msus also specifies that the directory path begins at the root. Remote msus is explained later in this section.

Subsequent slashes delimit individual names in the path.

Using " .. " in place of a directory name specifies the directory immediately superior to the current directory position. (Note that the root's superior directory is the root.) Using " . " in place of a directory name specifies the current directory position. To specify a file or directory subordinate to the current working directory, you do not include the current working directory in the directory path.

Examples

The directory path:

```
/USERS/BO/MANUAL_PLAN<mine*alone>
```

begins at the root.

The directory path:

```
../file1
```

begins at the directory immediately superior to the current working directory.

The directory path:

```
PROJECTS/WRITERS<writers_only>/samples:REMOTE
```

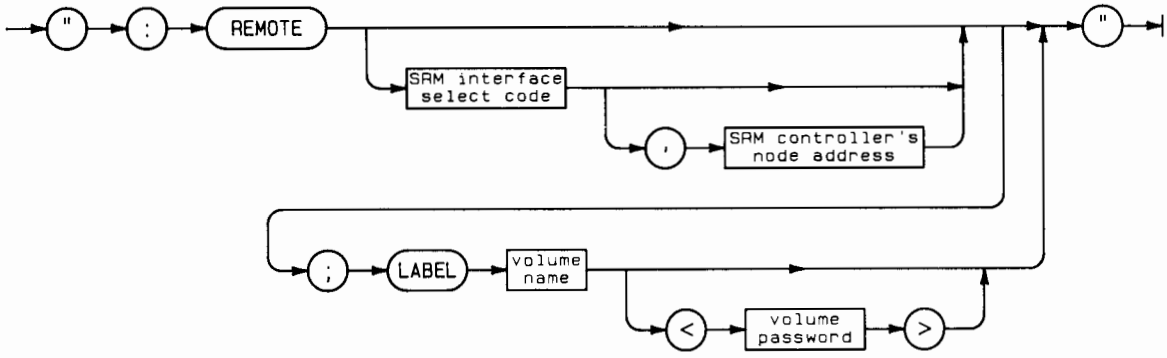
begins at the root.

The directory path:

```
dir_sub/file1
```

begins in the current working directory. In this example, `dir_sub` is immediately subordinate to the current working directory.

Remote msus



Item	Description/Default	Range Restrictions
SRM interface select code	integer constant	8 through 31
SRM controller's node address	integer constant	0 through 63
volume name	literal	any valid volume name (see Semantics)
volume password	literal	any valid password (see Semantics)

Semantics

The **volume name**, which is assigned at the volume's initialization, is used to identify a mass storage volume. Volume names consist of one to 16 characters, which may include uppercase and lowercase letters, digits 0 through 9, the underbar (_) character, the period (.) character, and national language characters (CHR$(161) through CHR$(254)).

A valid **volume password** consists of one to 16 characters, which may include any ASCII character except " > ".

The volume password allows complete access to all files on a mass storage volume, and is assigned when the volume is initialized. The volume password supercedes all access restrictions placed on files and directories by the PROTECT statement.

You need supply the **SRM interface select code** only if you wish to specify an SRM interface in your Series 200 workstation other than that identified by the default select code. If your workstation boots from the SRM, the default is the select code of the interface through which the boot ROM activates your workstation. If your workstation boots from a source other than SRM, the default select code is the lowest available SRM interface select code in the workstation. (The factory-set default value for the HP 98629A interface's select code is 21.)

The **SRM controller's node address** is necessary only if the node address of the controller is other than the default controller's node address.

To determine the defaults for your workstation use the following command sequence:

```
MSI ":REMOTE"    EXECUTE or Return
CAT    EXECUTE or Return
```

The header of the resulting catalog listing shows the default values for your workstation's SRM interface select code and SRM controller's node address, and the name of the default SRM system volume.

If you include the controller's node address, you must also include the SRM interface select code.

The LABEL secondary keyword identifies a volume, and is used mainly when more than one shared volume is on the SRM system. You need supply the volume label only if you are identifying a volume other than the default SRM system volume (in an SRM system having more than one shared disc) or if your application requires that you specify the volume password.

The Generic Remote msus

The generic msus syntax (not indicated in the syntax diagram above) bypasses the need for all information required by the remote msus syntax except the workstation's SRM interface select code. An example of this msus syntax is:

```
:,21
```

Examples

The remote msus:

```
:REMOTE
```

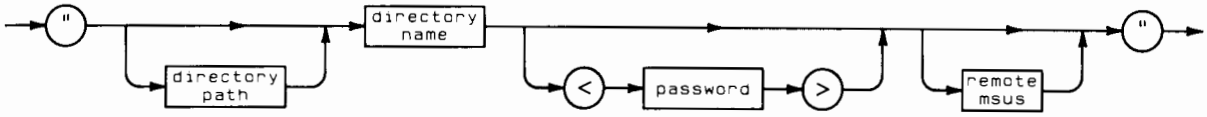
specifies the default SRM system volume.

The remote msus:

```
:REMOTE 21,1;LABEL VOL_TWO<secondpass>
```

specifies an SRM system volume. The LABEL syntax allows inclusion of the volume password in the remote msus. Note that, because the controller's node address is not the default and must be specified, the SRM interface select code must also be specified, even if that select code is the default.

Directory Specifier



Item	Description/Default	Range Restrictions
directory path	literal	(see diagram)
directory name	literal	any valid directory name (see Semantics)
remote msus	literal	(see diagram)

Semantics

A valid **directory name** consists of one to 16 characters, which may include uppercase and lowercase letters, digits 0 through 9, the underbar (_) character, the period (.) character, and national language characters (CHR$(161) through CHR$(254)). Spaces are ignored.

If no directory path is included, the current working directory (the directory specified in the latest MASS STORAGE IS statement) is assumed for the keyword's use. To specify a directory other than the current working directory, specify the directory path to the desired directory. (Refer to the syntax for directory path.) The directory path may begin at the current working directory or at the root.

The READ access capability for each directory included in the directory path must be public or the password that currently protects the READ capability must be included in the remote file specifier. A maximum of six directories may be included in the directory specifier. If the target directory is more than five directories away from the current working directory, move closer by changing the working directory (with MSI).

Examples

```
"/"
```

specifies the root. This form assumes that the SRM (remote mass storage) has previously been "entered" via some form of the `MSI ":REMOTE"` statement. (See directory path description.)

```
"../../../"
```

specifies the directory three levels superior to the current working directory. (See directory path description.)

```
",<MGR_pass>"
```

specifies the current working directory, with a password granting an access capability different from that currently in effect.

Access Capability Requirements

Because SRM allows password protection of files and directories, either certain access capabilities must be public or you must supply the password protecting those capabilities when you specify the file or directory in the keyword syntax. For more information on password protection and access capabilities, refer to the section on "Shared Access to Remote Directories and Files" earlier in this chapter and the PROTECT keyword entry in this reference.

The following chart lists BASIC keywords discussed in this section, indicating for each:

- whether the keyword is used with remote files, directories, or can be used with either;
- the access capabilities required on the directories superior to the specified directory or file;
- the access capabilities required on the specified directory or file itself.

Access requirements do not apply to the following keywords:

CHECKREAD
CONTROL
INITIALIZE
ON TIMEOUT
RESET
SCRATCH A
STATUS
UNLOCK
SYSTEM$

Note

For all keywords listed in the table, the READ capability must be public on all directories in the path to the target remote file or directory. Otherwise, you must supply the password protecting the READ capability on any such directory.

The entries in the following table indicate the access capabilities needed for use of the designated keyword. That is, the access capability listed must either be public (not protected with a password) or you must supply the password protecting the capability in the file or directory specifier included with the keyword.

For example, in an OUTPUT statement, if the WRITE capability on the file to which the data is to be written is not public, you must supply the password entitling you to write data to that file. (You would include the password as part of the remote file specifier in the statement assigning the I/O path name for the file to which the data is directed.) If the READ capability on the directory containing the remote file specified in the OUTPUT statement is not public, you must supply the appropriate password with the directory name in the directory path to the remote file.

Access Capabilities Required for Keyword Use

Keyword	Applies to	Access Capabilities Required	
		Directory/ File	Superior Directory
ASSIGN	file	at least 1	READ
CAT	either	READ	READ
COPY			
source	file	READ	READ
destination	file	–	READ & WRITE
CREATE ASCII	file	–	READ & WRITE
CREATE BDAT	file	–	READ & WRITE
CREATE DIR	directory	–	READ & WRITE
ENTER	file	READ	READ
GET	file	READ	READ
LOAD	file	READ	READ
LOADSUB	file	READ	READ
LOCK	file	at least 1	READ
MASS STORAGE IS	directory	–	READ
OUTPUT	file	WRITE	READ
PLOTTER IS	file	at least 1[1]	READ
PRINTER IS	file	at least 1[1]	READ
PROTECT	either	MANAGER	READ
PURGE	either	MANAGER	READ & WRITE
RENAME	either	MANAGER	READ & WRITE
RE-SAVE	file	READ & WRITE	READ & WRITE
RE-STORE	file	READ & WRITE	READ & WRITE
RE-STORE KEY	file	READ & WRITE	READ & WRITE
SAVE	file	–	READ & WRITE
STORE	file	–	READ & WRITE
STORE KEY	file	–	READ & WRITE
STORE SYSTEM	file	–	READ & WRITE
TRANSFER			
inbound	file	READ	READ
outbound	file	WRITE	READ

Dash (–) means "does not apply."

1 The statement, however, is not useful without WRITE access to the file.

Using Protected Files Created on a Pascal Workstation

The password protection assigned with the Pascal Filer's Access command imposes some restrictions on the use of BASIC keywords with a file protected with that command.

If a Pascal file's SEARCH capability alone is protected, the BASIC catalog listing will show the file's READ capability as public. The protection assigned for SEARCH, however, limits the types of BASIC read operations that can be performed on that file without the assigned password. For example, you can catalog a directory whose READ access capability is public and whose SEARCH access capability is not, but you cannot access any of the files or directories within that directory.

Similarly, the MANAGER access capability in BASIC encompasses the Pascal MANAGER, CREATELINK and PURGELINK capabilities.

BASIC vs. Pascal Protections

BASIC Access Capability	Equivalent Pascal Access Capability
MANAGER	MANAGER, CREATELINK, PURGELINK
READ	READ, SEARCH
WRITE	WRITE

Summary of BASIC Keyword Use on SRM

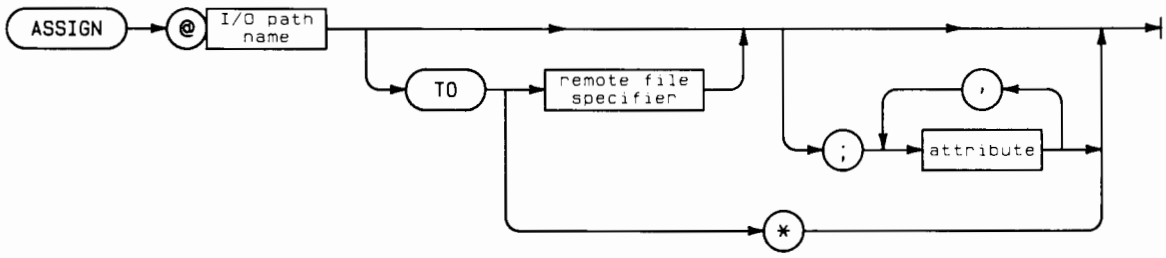
This section lists, in alphabetical order, the BASIC keywords that can be used with SRM and those that are unique to SRM (CREATE DIR, LOCK, PROTECT, the CAT PROTECT option, UNLOCK). Each keyword description in this section discusses only uses or features of the keyword that apply to its use on SRM.

Syntax diagrams appear only with those keywords requiring a different syntax for use with SRM. Where syntax diagrams are not included, you may follow the syntax described in the *BASIC Language Reference* manual, substituting **remote file specifier** syntax (described in the previous section) wherever "file specifier" is indicated in the keyword's syntax.

For access capability requirements, refer to the chart in the previous section.

ASSIGN

With SRM, I/O path names can be assigned to remote files, attributes can be assigned to the I/O path, and I/O paths can be closed. The following syntax and discussion describes only the use of ASSIGN with remote files. See the *BASIC Language Reference* manual for details of other uses of ASSIGN and the description of attributes associated with ASSIGN.



Example Statements

```
ASSIGN @Remote_file TO "DIR_JOHN/dir_proj/file1"
ASSIGN @File TO "P1/FredsData<pass>:REMOTE"
```

Semantics

Assigning an I/O path name to a remote file associates the I/O path with the file at the specified or default mass storage location.

ASSIGN opens any existing ASCII or BDAT file, regardless of protection on the file **except** when all access capabilities (MANAGER, READ and WRITE) are taken from the public. Attempts to use ASSIGN with a file whose capabilities are fully protected (without supplying the necessary passwords) result in Error 62.

In all other instances, a file's access capabilities are not checked at ASSIGN time. The specified operation on the file associated with the I/O path name is not executed, however, unless the file has the necessary access capability for that operation. For example, you may ASSIGN an I/O path name to a file that has only the READ capability public, but attempting to perform an OUTPUT operation without the password protecting the WRITE access capability generates Error 62.

ASSIGN does not create a file.

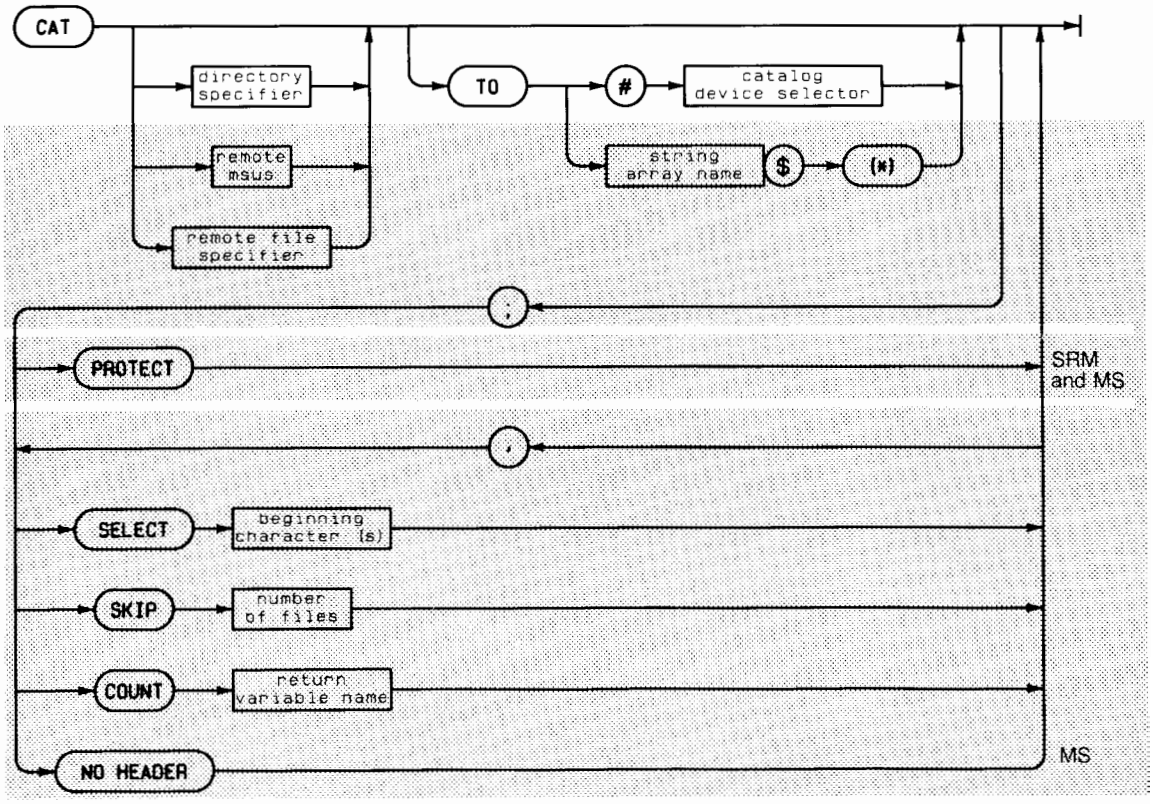
ASSIGN and Locked Files

Existing ASCII or BDAT files opened via ASSIGN are opened in shared mode, which means that several users can open a file at the same time. If you lock a file (refer to LOCK) and subsequently open that file via ASSIGN using the same @<*name*> (for example, to reset the file pointer), the ASSIGN automatically unlocks the file (refer to UNLOCK). To maintain sole access to the file, you must LOCK it again.

Closing an I/O path via ASSIGN (ASSIGN @...TO *) unlocks as well as closes the file (regardless of the number of LOCKs in effect for the file at the time).

CAT

With SRM, CAT lists all or specified portions of the contents of a directory or information regarding a specified PROG file. SRM adds the PROTECT option to the CAT statement. For a full description of the CAT statement syntax and CAT options, refer to the *BASIC Language Reference* manual.



Example Statements

```
CAT
CAT TO #701
CAT ":REMOTE"
CAT "../../.."
CAT "DIR1/DIR2"
CAT "A/B/C:REMOTE"
CAT "My_File";PROTECT
CAT ":REMOTE; LABEL Mastervol"
CAT;SELECT "D", SKIP Ten_files
CAT TO Directory$(*); NO HEADER
```

Semantics

To catalog remote directories, either you must include a remote msus in the CAT statement or the latest MASS STORAGE IS statement must have specified the desired remote msus. A catalog entry is listed for each file in the working or explicitly specified directory.

CAT to a Device

The catalog listing format used by the SRM system depends upon the line-width capacity of the device used for display.

When cataloging a remote directory on a 50-column display, the SRM system uses the following catalog format:

```
                                                          header
USERS/STEVE/PROJECTS/DIR1:REMOTE 21,0                     line 1
LABEL:     Disc1                                          line 2
FORMAT:    SDF                                            line 3
AVAILABLE SPACE:        54096                             line 4
                   PUB  FILE     NUMBER  RECORD  OPEN     line 5
FILE NAME          ACC  TYPE    RECORDS  LENGTH  STAT     line 6
================== ===  =====  =======  ======  ====
Common_data        MRW  ASCII        48     256  OPEN
Personal_data           BDAT        33     256  LOCK
Program_alpha       RW   PROG        44     256
HP9845_DATA          R   DATA?       22     256
HP9845_STORE       MRW  PROG?        9     256
Pascal_file.TEXT   MRW   TEXT       37     256
Program_500        MRW  PROG?       12     256
```

When cataloging a remote directory on an 80-column display, the SRM system uses the following catalog format:

```
                                                                                                  header
USERS/STEVE/PROJECTS/DIR1:REMOTE 21,0                                                             line 1
LABEL:     Disc1                                                                                  line 2
FORMAT:    SDF                                                                                    line 3
AVAILABLE SPACE:     54096                                                                        line 4
                        SYS  FILE   NUMBER   RECORD     MODIFIED      PUB OPEN                   line 5
FILE NAME          LEV  TYPE  TYPE  RECORDS   LENGTH DATE         TIME ACC STAT               line 6
================== ===  ==== ===== ======== ======== ============== === ====
Common_data          1       ASCII       48      256  2-Dec-83  13:20 MRW OPEN
Personal_data        1  98X6  BDAT        33      256  2-Dec-83  13:20     LOCK
Program_alpha        1  98X6  PROG        44      256  3-Dec-83  15: 6  RW
HP9845_DATA          1  9845  DATA        22      256 10-Oct-83   8:45  R
HP9845_STORE         1  9845  PROG         9      256 10-Oct-83   8:47 MRW
Pascal_file.TEXT     1  PSCL  TEXT        37      256 11-Nov-83  12:25 MRW
Program_500          1  9000  PROG        12      256 13-Dec-83   9:54 MRW
```

The *header* gives you the following information:

line 1 Directory name and remote msus. The full path to the specified directory is displayed. Passwords used in the path are not displayed.

If the directory path specifier contains more characters than the display width, the last 49 or 79 characters (depending on catalog format) in the path specifier are shown. An asterisk (*) as the leftmost character in the path specifier indicates that leading characters were truncated for the display.

The system remembers a maximum of 160 characters for any directory path specifier at a single time. If a path specifier contains more than 160 characters, the excess characters are removed from the beginning of the specifier and are not retained. This restriction does not affect movement within the directory structure.

line 2 — Volume label of the volume containing the directory.

line 3 — Directory format, such as SDF (Structured Directory Format). See your disc's operating manual for details.

line 4 — Number of bytes available on the volume (given in increments of 256 bytes).

lines 5 and 6 — Labels for columns of information given for each file. The information provided is summarized below.

The `FILE NAME` column lists the names of the remote files and directories in the directory.

The `LEV` column (80-column format only) shows the level of the file relative to the current working directory or specified directory. The level is always shown as `1` in directory listings for Series 200 workstations.

The `PUB ACC` column lists the access capabilities available to all SRM system users. The three capabilities are READ, (`R`) WRITE (`W`) and MANAGER (`M`).

- Public MANAGER capability on a file or directory allows any user on the SRM system to PURGE that file or directory and to modify or add to its passwords (with PROTECT). Password-protected MANAGER capability gives users who supply the required password both READ and WRITE capabilities as well as MANAGER capability.
- READ capability on a directory allows you to access any file or directory in the directory. The READ capability on a file allows you to read the contents of the file.
- WRITE capability on a directory allows you to create or delete a file or directory in that directory. The WRITE capability on a file allows you to write information into that file.

The `SYS TYPE` column (80-column format only) shows the type of system used to create the file. The system type is not shown for ASCII files and directories. `98XG` denotes a Series 200 computer. If the SRM system does not recognize the system type, a coded identifier, obtained from the system being identified, appears in this column.

The `FILE TYPE` column indicates the file's type. Directories are indicated as type `DIR`. In the 50-column format, a question mark is appended to the file type if the file was not created on a Series 200 computer and was a type other than ASCII or DIR. For example, in the display illustrated earlier, DATA and PROG files created on an HP 9845 are listed as such, but shown with the question mark.

File types recognized by the BASIC system on SRM are: ASCII, BDAT, BIN, DIR, PROG, and SYSTM, as well as Series 200 Pascal and Series 500 file types.

If the system does not recognize a file's type, a coded file type identifier, obtained from the system originating the file, appears in the `FILE TYPE` column.

The NUMBER OF RECORDS column indicates the number of records in the file and the RECORD LENGTH column indicates the number of bytes constituting each of the file's records.

The MODIFIED columns (80-column format only) show the date and time the file's contents were last changed.

The OPEN STAT column shows whether the file is currently open (OPEN), locked (LOCK) or corrupt (CORR). OPEN indicates that the file has been opened, via ASSIGN, by a user. An open file is available for access from other workstations. LOCK means the file is accessible only from the workstation at which the file was locked. CORR indicates that the disc lost power while accessing the file, possibly altering the file's contents. If the entry is blank, the file is closed and available to any user.

Note

If a file's status is shown as corrupt (CORR), you should run the DSCK Utility program to check the directory structure and its integrity on the SRM system disc. Refer to the *SRM Operating System Manual* for details.

CAT to a String Array

Regardless of the workstation's display width, a CAT to a string array always produces the 80-column format.

The PROTECT Option

PROTECT is a CAT option provided by the SRM BIN file and available only on SRM. This option also requires the MS BIN file. The PROTECT option displays the password(s) and associated access capabilities for the specified file or directory.

For example, the statement:

```
CAT "Test_file<MPASS>:REMOTE";PROTECT
```

might produce the display:

```
PASSWORD             CAPABILITY
================ ==================
MPASS            MANAGER,READ,WRITE
WPASS            WRITE
RPASS            READ
PASSWORD         MANAGER
```

Use of this option requires MANAGER access capability on the file or directory. If the MANAGER capability is public, the PROTECT option may be used by any SRM user.

PROTECT must be specified separately from other CAT options, and is allowed only with SRM files and directories. Using PROTECT with media other than SRM results in ERROR 1 Configuration error.

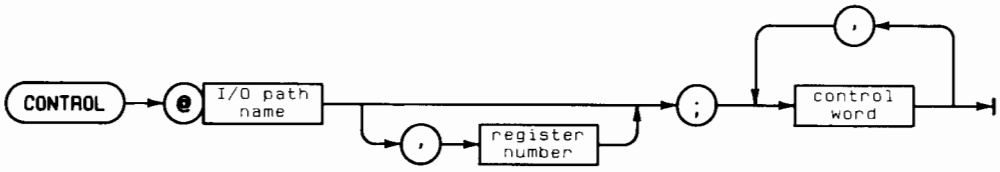
CHECKREAD

For SRM, CHECKREAD is implemented as a no-op, because the CHECKREAD function is already performed for every read and write statement on the SRM. Further checking places overhead on the system and doing so would not be accurate. With SRM, CHECKREAD **may or may not** cause a true write to the disc, while its read would probably only access the buffers in the SRM system. SRM's internal read and write checking and the automatic checking on the link make using CHECKREAD unnecessary.

CONTROL

With SRM, CONTROL sends control information to the internal table associated with an I/O path name assigned to an ASCII or BDAT file (see ASSIGN). Refer to the CONTROL keyword entry in the *BASIC Language Reference* manual for a full explanation of CONTROL syntax.

Control registers are listed in the "I/O Path Status and Control Registers" table in the Interface Registers section of the *BASIC Language Reference* manual.

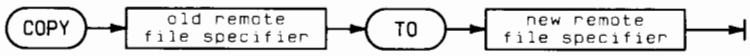


Example Statement

```
CONTROL @Rand_file,7;File_length
```

COPY

With SRM, COPY allows copying of individual remote files. Remote directories and volumes cannot be copied.



Example Statements

```
COPY "/Dir_1/File_1" TO "Dir_3/File_1"
COPY "File:INTERNAL" TO "File:REMOTE 21,0"
COPY Dir_path$&File$&Msus& TO "File:INTERNAL"
```

Semantics

The contents of the old remote file are copied to the new remote file and an entry is placed in the destination directory. The old and new remote files may be in the same directory, but the new remote file's name must be unique.

Although you may include a password in the new remote file specifier, the system ignores the password. If you wish to protect access to the new file, you must assign the password with PROTECT.

CREATE ASCII

With SRM, CREATE ASCII creates a new remote ASCII file, placing a corresponding directory entry in the current working directory or specified remote directory.

Example Statements

```
CREATE ASCII "Text03", 100
CREATE ASCII "/Dir1/Dir2/ASCIIFILE",25
```

Semantics

The name of the newly-created ASCII file must be unique within its containing directory.

CREATE ASCII **does not** open the file. Files are opened with the ASSIGN statement. If an error occurs during execution of CREATE ASCII, no directory entry is made and the file is not created.

The specified number of records determines the number of physical records for a remote ASCII file's initial space allocation. The physical records of an ASCII file have a fixed length of 256 bytes. (Logical records have variable lengths, determined automatically when an OUTPUT, SAVE or RE-SAVE statement is used.)

Storage space for subsequent saving of remote files is allocated only when needed. When data is added to a remote file such that saving the modified file would overflow the file's current space allocation, the SRM system adds another extent. An extent is a space allocation whose size is determined by multiplying the specified number of records by the record size.

When the remote file is created, all access capabilities are public. Including a password in the CREATE ASCII statement's remote file specifier does not protect the file. You must use PROTECT to assign passwords. You will not receive an error message for including a password, but passwords in the CREATE ASCII statement are ignored.

CREATE BDAT

With SRM, CREATE BDAT creates a new remote BDAT file, placing a corresponding directory entry in the current working directory or specified remote directory.

Example Statements

```
CREATE BDAT "File",Records,Rec_size
CREATE BDAT "/Dir1/Dir2/BDATFILE",25,128
CREATE BDAT "Dir/File:REMOTE",10
```

Semantics

The name of the newly-created BDAT file must be unique within its containing directory.

CREATE BDAT **does not** open the file. Files are opened with the ASSIGN statement. If an error occurs during execution of CREATE BDAT, no directory entry is made and the file is not created.

The specified number of records determines the number of physical records for a remote BDAT file's initial space allocation. The length of a BDAT file's physical records is either specified by the record size parameter or set to 256 bytes if no record size is specified.

Storage space for subsequent saving of remote files is allocated only when needed. When data is added to a remote file such that saving the modified file would overflow the file's current space allocation, the SRM system adds another extent. An extent is a space allocation whose size is determined by multiplying the specified number of records by the record size. On SRM, CREATE BDAT does not allocate a sector for system use, as it does with local files.

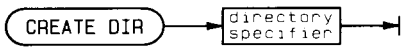
When the remote file is created, all access capabilities are public. Including a password in the CREATE BDAT statement's remote file specifier does not protect the file. You must use PROTECT to assign passwords. You will not receive an error message for including a password, but passwords in the CREATE BDAT statement are ignored.

The data in remote BDAT files can be accessed both serially and randomly.

CREATE DIR

Option Required	SRM,DCOMM
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement creates a directory in either the current working directory or in the specified remote directory of an SRM mass storage device.



Example Statements

```
CREATE DIR "Under_work_dir"
CREATE DIR "Level1/Level2/New_dir:REMOTE 21,3"
CREATE DIR "/Level1/Level2/New_dir"
CREATE DIR "Level1<RWpassword>/New_dir"
```

Semantics

This statement creates a special 24-byte file of type DIR and a corresponding directory entry in the current working directory or specified remote directory. The DIR file, or directory, keeps information on files and directories immediately subordinate to itself.

The name of the newly-created directory must be unique within its containing directory.

Like remote data files, DIR files are extensible. Extents are added in 24-byte increments. As each directory or data file is created within a directory, a 24-byte record identifying the addition is added to the DIR file.

If no directory path is included in the directory specifier, the directory is created within the current working directory (the directory specified in the latest MASS STORAGE IS statement). To specify a target directory other than the current working directory, specify the directory path to the desired directory.

You cannot assign passwords to a directory when you create it. Passwords are assigned only via PROTECT. If an error occurs during execution of CREATE DIR, the directory entry in the superior directory is not made, and the directory is not created.

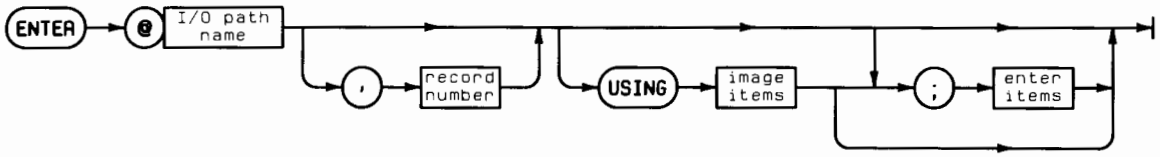
DIR files are opened with the MASS STORAGE IS (MSI) statement.

Refer to the section on "Syntax for Remote File and Directory Specification" earlier in this chapter for details on the semantics of directory specifiers.

ENTER

With SRM, ENTER is used to read data from a remote data file identified by an I/O path name and to assign the value(s) to variable(s). (See also ASSIGN.)

The capabilities available for using ENTER with remote files are the same as those for using ENTER with local files. Refer to the ENTER keyword entry in the *BASIC Language Reference* manual for a full explanation of ENTER syntax.



Example Statements

```
ENTER @Remote_file,REC;Alpha$,Beta$,Gamma$
ENTER @Name_of;A,B
```

Semantics

Entering data from remote files requires the READ access capability on the superior directory and on the file from which the data are to be read. If this capability is not public or if a password protecting this capability was not used at the time the file was ASSIGNed, an error is reported.

GET

With SRM, GET reads the specified remote ASCII file and attempts to store the data in memory as program lines.

Example Statements

```
GET "Filename:REMOTE"
GET "/Dir1/Dir2/Dir3/filename<READPass>"
```

Semantics

You may use GET with any ASCII file whose data is in the format of a BASIC program (that is, having numbered lines). Although you may also use GET with ASCII files created on non-Series 200 SRM workstations (HP 9835, HP 9845 or Model 520), any line that is not valid BASIC syntax for Series 200 computers is stored as a commented (!) program line.

When used on SRM, GET is executed in shared mode, which means that several users can get one file at the same time. Attempts to get a locked file (see LOCK) result in Error 453. Additionally, you cannot get a file while it is being saved. The SAVE and RE-SAVE operations open the file in exclusive mode (shown as LOCK in a CAT listing) and enforce that status until the SAVE or RE-SAVE is complete. While in exclusive mode, the file is accessible only to the SRM workstation executing the SAVE or RE-SAVE.

INITIALIZE

With SRM, INITIALIZE can be executed from the controller only. You cannot use INITIALIZE from a workstation on SRM.

LOAD

With SRM, LOAD loads the contents of remote PROG or BIN files into memory, or sets the typing-aid definitions of the softkeys according to the contents of a remote BDAT file.

Example Statements

```
LOAD "Program_2"
LOAD "/Dir1/Dir2/Prog2",500
LOAD "Dir3/Prog_1:REMOTE"

LOAD BIN Dir$&File$&Msus$
LOAD BIN "dir1/dir2/bin_file<Readpass>:REMOTE 21,5;LABEL Disc"

LOAD KEY "KEYS:REMOTE"
LOAD KEY "/Dir1/Dir2/Keyfile"
```

Semantics

LOAD

LOAD can be used with remote PROG files (created with the STORE statement). LOAD is executed in shared mode, which means that several users can load a file at the same time. Files being stored with the STORE or RE-STORE statements are locked during that operation and cannot be accessed for loading.

LOAD BIN

LOAD BIN can be used with remote BIN files. LOAD BIN is executed in shared mode, which means that several users can load a BIN file at the same time.

BIN files can be loaded into a workstation from the SRM without the SRM BIN file present in the workstation. Refer to the "Booting from SRM" section of this chapter for more details.

LOAD KEY

LOAD KEY can be used with remote BDAT files (created with the STORE KEY statement). LOAD KEY is executed in shared mode, which means that several users can perform a LOAD KEY from a BDAT file at the same time. Files being stored with the STORE KEY or RE-STORE KEY statements are locked during that operation and cannot be accessed for loading.

LOADSUB

With SRM, LOADSUB allows you to load subprograms from a remote PROG file into your workstation.

Example Statements

```
LOADSUB FROM "APSUBS"
LOADSUB FNReplace$ FROM "SUBFILE"
LOADSUB ALL FROM Subfile$
LOADSUB ALL FROM "Dir3/Prog file<ReadPass>"
LOADSUB ALL FROM "/Dir1/Dir2/Prog23"
```

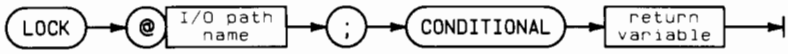
Semantics

With SRM, LOADSUB is executed in shared mode, which means that several workstations can perform a LOADSUB of a file at the same time. PROG files being stored with the STORE or RE-STORE statement are locked during that operation and cannot be accessed for loading.

LOCK

Option Required	SRM,DCOMM
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement prevents other SRM workstations from accessing the remote file to which the I/O path name is currently assigned (see ASSIGN).



Item	Description/Default	Range Restrictions
I/O path name	name identifying an I/O path	any valid name (See Glossary.)
return variable	name of a numeric variable	any valid name (See Glossary.)

Example Statements

```
LOCK @File;CONDITIONAL Result
LOCK @Ascii_1;CONDITIONAL Error_number
```

Semantics

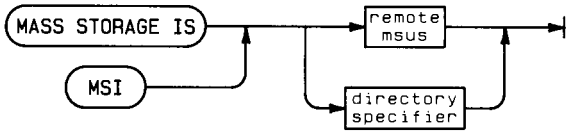
This statement establishes sole access to a file that has been opened with an ASSIGN statement. This exclusive access remains assigned to the workstation executing the LOCK statement until an UNLOCK statement is executed by that workstation. The UNLOCK function is also a result of SCRATCH A, (RESET) and ASSIGN...TO * (explicitly closing an I/O path).

A file may be locked several times. The system counts the number of LOCKs on a file, and an equal number of UNLOCKs must be executed to unlock the file. When an I/O path name is closed (for example, by ASSIGN...TO *), **all** LOCKs of that I/O path name are cleared.

If the LOCK is successful, the value of the return variable will be zero. Otherwise, the return variable's value will be the error number corresponding to the cause of the LOCK's failure.

MASS STORAGE IS

With SRM, MASS STORAGE IS specifies the SRM working directory.



Example Statements

```
MSI "Dir1/Dir2/Project_dir"
MSI ".."
MASS STORAGE IS ":<password>"
MSI ":REMOTE"
```

Semantics

SRM allows directories or volumes to be assigned as system mass storage. If you specify the volume password in an MSI statement, that password is automatically applied to all accesses that use the default msus (that is, no remote msus is specified in the remote file specifier) until a remote msus is included in a subsequent MSI.

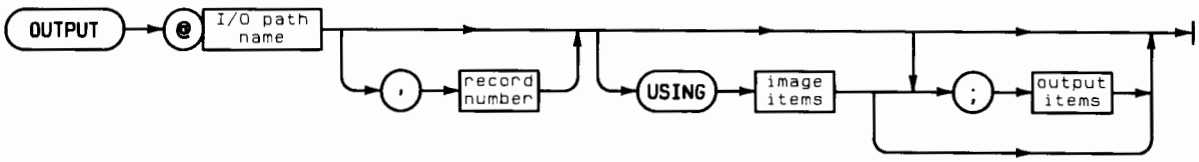
ON TIMEOUT

With SRM, ON TIMEOUT defines and enables a branch resulting from an I/O timeout on the specified SRM interface. Although ON TIMEOUT is supported on SRM, **its use should be avoided** because the asynchronous nature of the SRM system does not allow predictable results.

A TIMEOUT occurring during statements such as RE-SAVE and RE-STORE may leave a temporary file on the mass storage device. The file's name is a 10-character identifier (the first character is an alpha character, the rest are digits) derived from the value of the workstation's real-time clock when the TIMEOUT occurred. You may wish to check the contents of any such file before purging.

OUTPUT

With SRM, OUTPUT writes item(s) to the remote file to which the specified I/O path name is assigned (see ASSIGN). Refer to the OUTPUT keyword entry in the *BASIC Language Reference* manual for a full explanation of OUTPUT syntax.



Example Statement

```
OUTPUT @File;Array(*),END
```

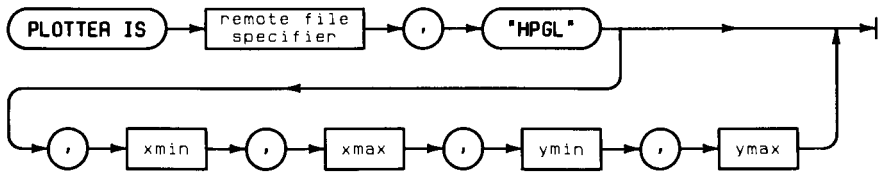
Semantics

You must have WRITE access capability on the remote file to output data to the file.If this capability is not public or if a password protecting this capability was not used at the time the file was ASSIGNed, Error 62 is reported.

If the data output to the file with this statement would overflow the file's space allocation, the system allocates the additional space needed to save the file (provided the disc contains enough unused storage space). Refer to the "System Concepts" section of this chapter for more details on the extensible nature of remote files.

PLOTTER IS

With SRM, PLOTTER IS causes all subsequent plotter output to go to the specified remote BDAT file. Refer to the PLOTTER IS keyword entry in the *BASIC Language Reference* manual for a full explanation of PLOTTER IS syntax.



Example Statements

```
PLOTTER IS "/PL/Plotfile"
PLOTTER IS "Plotfile:REMOTE",,"HPGL",6,25,256,25,6,975,186,975
```

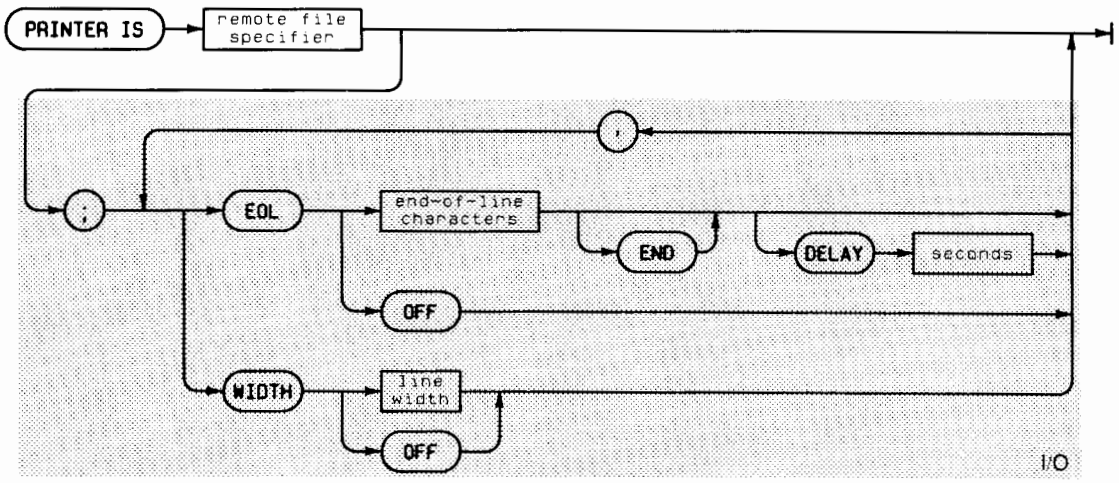
Semantics

If the specified remote file is in the SRM plotter spooler directory and the file contains data, when the file is closed the SRM system sends the data to the plotting device and then purges the file. You may close the file by executing another PLOTTER IS statement, SCRATCH A or SCRATCH BIN, or by pressing (RESET).

No end-of-file error occurs on SRM. If the data output to the file with this statement would overflow the file's space allocation, the system allocates the additional space needed to save the file (provided the disc contains enough unused storage space). Refer to the "System Concepts" section of this chapter for more details on the extensible nature of remote files.

PRINTER IS

With SRM, PRINTER IS specifies a remote BDAT file as the system printing file. Refer to the PRINTER IS keyword entry in the *BASIC Language Reference* manual for a full explanation of PRINTER IS syntax.



Example Statements

```
PRINTER IS "Spooler:REMOTE"
PRINTER IS "My_dir/Temp_print";WIDTH 80
```

Semantics

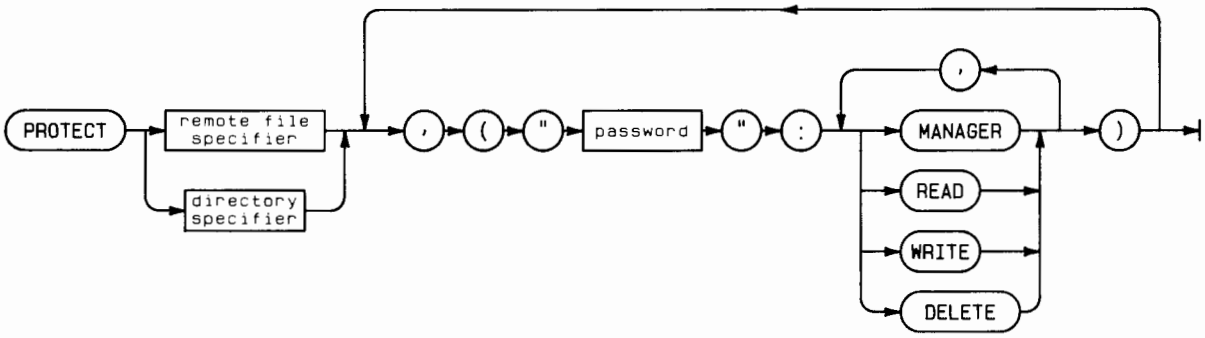
The system printing file receives all data sent by the PRINT statement, all data sent by CAT and LIST statements in which the destination is not explicitly specified, and other output generated by the BASIC system.

If the specified remote file is in the SRM printer spooler directory and the file contains data, when the file is closed, the SRM system sends the data to the printer and then purges the file. You may close the file by executing another PRINTER IS statement, or a SCRATCH A or SCRATCH BIN command.

No end-of-file error occurs on SRM. If the data output to the file with this statement would overflow the file's space allocation, the system allocates the additional space needed to save the file (provided the media contains enough unused storage space). Refer to the "System Concepts" section of this chapter for more details on the extensible nature of remote files.

PROTECT

With SRM, this statement protects access capabilities by assigning passwords to remote files and directories. The use of PROTECT with SRM is distinct from its use with local files (described in the *BASIC Language Reference* manual).



Example Statements

```
PROTECT "dir:REMOTE",("mgr":MANAGER),("rw":READ,WRITE)
PROTECT "File<rw>",("rw":DELETE)
```

Semantics

PROTECT allows you to control access to remote files and directories by protecting access capabilities with password(s). Access capabilities are either public (available to all SRM users) or password-protected (available only to users supplying the correct password with the file or directory specifier).

The three access capabilities – MANAGER, READ and WRITE – are public unless the PROTECT statement associates a password with one or more of those capabilities.

Once the capability on a given file or directory is password-protected, the capability can be exercised on the file or directory only if the correct password is included in the file or directory specifier. For instance, if a file's READ capabilities are protected, any user wishing to execute a command or statement that reads the file must supply a password protecting the file's READ capability.

MANAGER

Public MANAGER capability allows any SRM user to PROTECT, PURGE or RENAME a file or directory. Password-protected MANAGER capability provides READ and WRITE, as well as MANAGER, access capabilities to users who know the password.

You must have MANAGER capabilities on a file or directory to PROTECT the access capabilities on that file or directory. This includes adding, deleting and changing passwords.

READ

READ capability on a file allows use of commands and statements that read the contents of a file (for example: ENTER, LOAD, GET). READ capability on a directory allows you to read the files in the directory (CAT), or to "pass through" a directory by including the directory name (and password, if assigned) in a directory path.

WRITE
WRITE capability on a file allows use of commands and statements that write to the file (for example: OUTPUT, RE-SAVE, RE-STORE). WRITE capability on a directory allows use of commands that add or delete file names in the directory (for example: SAVE, STORE, PURGE, CREATE, RENAME).

Use of PROTECT
Each PROTECT statement allows up to six password/capability combinations per statement. The number of PROTECT statements that can be executed for each file or directory is unlimited, however, as long as each password is unique.

Successive associations of capabilities with the same password are not cumulative. To retain previous capability assignments for a file or directory, you must include those assignments in subsequent PROTECT statements designating the same password for that file or directory.

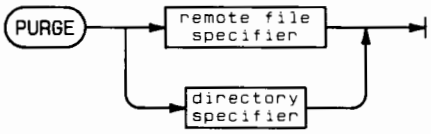
For example, say you protected the READ access capability on a file with the password *passme* then wanted to change that assignment so that *passme* would protect both the READ and WRITE access capabilities for that file. If you executed a second PROTECT statement associating *passme* with the WRITE capability only, *passme* would no longer protect the READ capability. Instead, you should specify the password and **both** the READ and WRITE capabilities in the second PROTECT statement.

To modify the access capabilities protected by a password, execute the PROTECT with the existing password and the new password/capability pair(s).

The secondary keyword DELETE is used to delete existing password assignments for a file or directory. To be effective, DELETE must be the only secondary keyword used with a password/capability pair in the PROTECT statement. Otherwise, DELETE is ignored. MANAGER capability is required to perform the DELETE. A DELETE executed without MANAGER capability results in a protect code violation error.

PURGE

With SRM, PURGE deletes a file entry from a directory or an empty remote directory from its superior directory.



Example Statements

```
PURGE "File"
PURGE "Dir_a<RWpass>/File<MGRpass>"
PURGE "Dir1/Dir2/Dir3"
```

Semantics

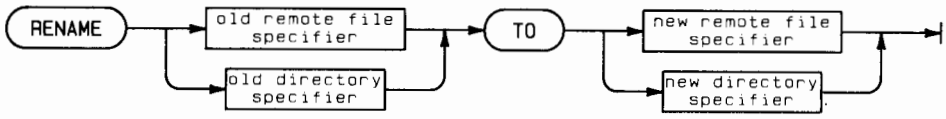
Only remote files and directories that are closed can be purged. Remote files are closed by ASSIGN...TO * (explicitly closes an I/O path). SCRATCH A closes both directories and remote files. All remote files except those opened with the PRINTER IS statement are also closed by (RESET). The current working directory is closed by an MSI to a different directory.

Once a file or directory is purged, its contents cannot be recovered.

To be purged, directories must be empty (must not contain any subordinate files or directories) as well as closed.

RENAME

With SRM, RENAME changes a remote file's name in a remote directory and performs limited file relocation.



Example Statements

```
RENAME "Old_name" TO "New_name"
RENAME "Dir1<RWpass>/F1<MGRpass>" TO "Dir2<RWpass>/F1"
RENAME "THIS:REMOTE" TO "THAT"
```

Semantics

RENAME can be used to change the name of remote files and directories or to move files within the directory structure. Directories cannot be moved with RENAME. Moving of files must occur within a single volume. If you move a file with RENAME, the original file ("old remote file specifier") is purged.

A maximum of nine names (file or directory) are allowed in the combined file/directory specifiers in the RENAME statement. No more than six names are allowed in either specifier individually. (The number of names in the old file/directory specifier plus the number of names in the new file/directory specifier must not exceed nine.)

Files and directories must be closed before being renamed. Remote files are closed by ASSIGN...TO * (explicitly closes an I/O path). SCRATCH A closes both directories and remote files. All remote files except those opened with the PRINTER IS statement are also closed by (RESET). The current working directory is closed by an MSI to a different directory.

Existing passwords are retained by the renamed file or directory. The new file name must not duplicate the name of any existing file in the destination directory.

RE-SAVE

With SRM, RE-SAVE creates a remote ASCII file and copies program lines as strings into that file.

Example Statements

```
RE-SAVE "File"
RE-SAVE "Dir<RWpass>/File<RWpass>"
```

Semantics

RE-SAVE opens the remote file in exclusive mode (denoted as LOCK in a CAT listing) and enforces that status on the file until the RE-SAVE is complete. While in exclusive mode, the file is inaccessible to all SRM workstations other than the one executing the RE-SAVE.

If the file does not already exist, RE-SAVE performs the same action as SAVE. Including a password in the RE-SAVE statement's remote file specifier does not protect the file. Passwords are assigned only with PROTECT. You do not receive an error for including a password with the specifier of a remote file that does not already exist, but the system ignores the password.

Passwords assigned to an existing file are retained when a RE-SAVE is performed on the file. If you specify the wrong password on a protected file, the system returns an error message.

Use of RE-SAVE on SRM may leave temporary files on the mass storage media if (CLR I/O) or (RESET) is pressed or a TIMEOUT occurs during the RE-SAVE. The file name of the temporary file is a 10-character name (the first is an alpha character, others are digits) derived from the value of the workstation's real-time clock when the interruption occurred. You may wish to check the contents of any such file before purging.

RESET

With SRM, this statement resets the pointers of a remote file identified by an I/O path name (see ASSIGN).

Example Statement

```
RESET @Remote_file
```

RE-STORE

With SRM, RE-STORE creates a remote file and stores the BASIC program or typing-aid key definitions in that file.

Example Statements

```
RE-STORE "Prog_a"
RE-STORE "Dir<RWpass>/Prog_z<RWpass>"
RE-STORE KEY "KEYS:REMOTE"
RE-STORE KEY "TYPING"
```

Semantics

RE-STORE creates a remote PROG file, and RE-STORE KEY creates a remote BDAT file.

RE-STORE opens the remote file in exclusive mode (denoted as LOCK in a CAT listing) and enforces that status on the file until the RE-STORE is complete. While in exclusive mode, the file is inaccessible to all SRM workstations other than the one executing the RE-STORE.

If the file does not already exist, RE-STORE performs the same action as STORE. Including a password in the RE-STORE statement's remote file specifier does not protect the file. Passwords are assigned only with PROTECT. You do not receive an error for including a password with the specifier of a remote file that does not already exist, but the system ignores the password.

Passwords assigned to an existing file are retained when a RE-STORE is performed on the file. If you specify the wrong password on a protected file, the system returns an error message.

Use of RE-STORE on SRM may leave temporary files on the mass storage media if (CLR I/O) or (RESET) is pressed or a TIMEOUT occurs during the RE-STORE. The file name of the temporary file is a 10-character name (the first is an alpha character, others are digits) derived from the value of the workstation's real-time clock when the interruption occurred. You may wish to check the contents of any such file before purging.

SAVE

With SRM, SAVE creates a remote ASCII file and copies program lines as strings into the file.

Example Statements

```
SAVE "THE_WHALES"
SAVE "Dir<RWpass>/File"
SAVE "Ascii_file:REMOTE"
```

Semantics

SAVE opens the remote file in exclusive mode (denoted as LOCK in a CAT listing) and enforces that status on the file until the SAVE is complete. While in exclusive mode, the file is inaccessible to all SRM workstations other than the one executing the SAVE.

Including a password in the SAVE statement's remote file specifier does not protect the file. Passwords are assigned only with PROTECT. You do not receive an error for including a password with the remote file specifier, but the system ignores the password.

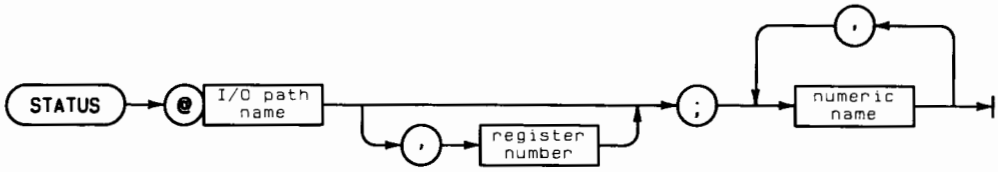
SCRATCH A

With SRM, SCRATCH A releases the system resources allocated to the workstation executing the SCRATCH A, making those resources available to other SRM workstations. SCRATCH A closes all files and directories, and resets the workstation's working directory to the default msus (the mass storage unit from which the workstation booted).

If the workstation has Boot ROM version 3.0 or later, and booted from the SRM, SCRATCH A resets the working directory to the root of the default system volume. If the workstation has an earlier version boot ROM or Boot ROM 3.0L, SCRATCH A resets the working directory to the device from which the workstation booted (for example, :INTERNAL if the workstation booted from a built-in drive).

STATUS

With SRM, STATUS returns the contents of I/O path name status registers (see ASSIGN). Refer to the STATUS keyword entry in the *BASIC Language Reference* manual for a full explanation of STATUS syntax. Status registers are listed in the "I/O Path Status and Control Registers" table in the Interface Registers section of the *BASIC Language Reference* manual.



Example Statement

```
STATUS @File,5;Record
```

STORE

With SRM, STORE creates a remote file and stores a program or typing-aid key definitions into it.

Example Statements

```
STORE "Prog32"
STORE "Dir<RWpass>/Program"
STORE KEY "KEYS:REMOTE"
STORE KEY "/USERS/KRIS/TYPING"
```

Semantics

STORE creates a remote PROG file, and STORE KEY creates a remote BDAT file.

STORE opens the remote file in exclusive mode (denoted as LOCK in a CAT listing) and enforces that status on the file until the STORE is complete. While in exclusive mode, the file is inaccessible to all SRM workstations other than the one executing the STORE.

Including a password in the STORE statement's remote file specifier does not protect the file. Passwords are assigned only with PROTECT. You do not receive an error for including a password with the remote file specifier, but the system ignores the password.

STORE SYSTEM

With SRM, STORE SYSTEM stores the entire BASIC operating system currently in memory (including any BIN files) into the specified remote file.

Example Statements

```
STORE SYSTEM "SYSTEM_B1:REMOTE"
STORE SYSTEM "/SYSTEMS/SYSTEM_NEW"
```

Semantics

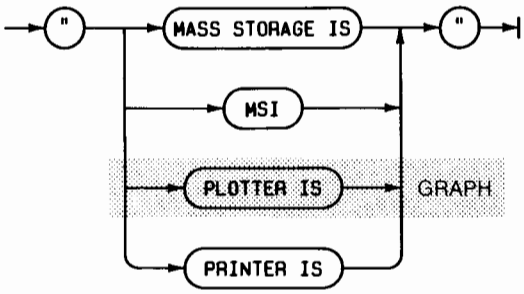
Including a password in the STORE SYSTEM statement's remote file specifier does not protect the file. Passwords are assigned only with PROTECT. You do not receive an error for including a password with the remote file specifier, but the system ignores the password.

The READ access capability on the system file created with STORE SYSTEM must be public to allow use of the file for booting.

SYSTEM$

With SRM, this function returns a string containing system status and configuration information.

literal form of type of information:



Example Statement

```
SYSTEM$("MSI")
SYSTEM$("PRINTER IS")
SYSTEM$("PLOTTER IS")
```

Semantics

The system configuration information returned when SYSTEM$ is executed on SRM includes the full remote file specifier describing the file or directory about which the information is requested. Passwords in the specifier are not included.

The system remembers a maximum of 160 characters for any one specifier. If a specifier contains more than 160 characters, the excess characters are removed from the beginning of the specifier and are not retained. An asterisk (*) as the leftmost character in the specifier indicates that leading characters were truncated for the function.

TRANSFER

With SRM, this statement initiates unformatted data transfers between the workstation and remote mass storage devices. Either the source or destination of the transfer is specified as an I/O path name assigned to a remote BDAT file (see ASSIGN). Refer to the TRANSFER keyword entry in the *BASIC Language Reference* manual for a full explanation of TRANSFER syntax.

Example Statements

```
TRANSFER @Buffer TO @File;CONT
TRANSFER @Dir_Path TO @Destination;COUNT 256
TRANSFER @Source TO @Buffer;DELIM "/"
TRANSFER @Path TO @Buffer;RECORDS 12,EOR(COUNT 8)
```

Semantics

TRANSFER behaves the same on SRM as with local mass storage, except that inbound and outbound transfer execution is not overlapped. Whereas the discs on the SRM may be cabable of overlapped operation, the SRM system performs TRANSFERs serially. This difference only matters in applications, such as data logging, where you may want a program to be able to execute other statements before the transfer has completed. For further details, refer to the "Transfer Performance" section in the "Advanced Transfer Techniques" chapter of the *BASIC Interfacing Techniques* manual.

UNLOCK

Option Required	SRM,DCOMM
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement is used to remove exclusive access (placed by the LOCK statement) to a remote file identified by an I/O path name (see ASSIGN).

Item	Description/Default	Range Restrictions
I/O path name	name identifying an I/O path to a remote file	any valid name (See Glossary.)

Example Statements

```
UNLOCK @File
IF Done THEN UNLOCK @File
```

Semantics

This statement unlocks a file previously locked with the LOCK statement. While a file is locked, other SRM workstations cannot access the file. After UNLOCK, other users may access the file provided they possess the proper access capability (or capabilities).

If multiple LOCKs were executed on the file, the same number of UNLOCKs must be executed to unlock the file.

UNLOCK is performed automatically by SCRATCH A, (RESET) and ASSIGN...TO * (explicit closing of an I/O path).

SRM BASIC Error Codes for HP Series 200 Computers

450	Volume not found
451	Volume labels do not match
453	File is use
454	Directory formats do not match
455	Possibly corrupt file
456	Unsupported directory operation
457	Passwords not supported
458	Unsupported directory format
459	Specified file is not a directory
460	Directory not empty
461	Duplicate passwords not allowed
462	Invalid password
465	Invalid rename across volumes
466	Duplicate volume entries
481	File locked or open exclusively
482	Cannot move a directory via a RENAME
483	System down
484	Password not found
485	Invalid volume copy

HP Series 200 Pascal Workstation Use on SRM

Chapter 2

The Shared Resource Management system (SRM) connects several workstations (computers) to form a network that allows sharing of files and peripherals. This network is controlled by a system controller.

The Pascal language system incorporates features to access shared resources. This chapter outlines the use of an HP Series 200 computer operating under Pascal as an SRM workstation.

Note

This chapter summarizes Pascal language features that support the use of SRM and does not include a comprehensive discussion of Pascal language system terms and concepts.

For detailed information, refer to the *Pascal User's Guide* and the *Pascal Workstation System* manual (previously the *Pascal User's Manual*).

System Concepts

How Your Workstation Connects to SRM

To use the SRM system, you need to know how to move information from one point to another in the system. Everything in the SRM system has a name and/or address. This section is a guided tour of the SRM configuration.

Your Workstation's Identification

Your workstation's contact with the SRM system is through an HP 98629A (Resource Management Interface) card. This card is placed in one of the Input/Output slots in the rear of your computer.

Your workstation is uniquely identified to the SRM system by its node address, assigned when the workstation was added to the SRM system. The node address is set on the HP 98629A card by means of switches on the card.

The HP 98629A card also has a setting called the select code, which should normally be left at 21. All of the software shipped to you that uses your workstation's select code expects to see 21. See the *SRM Hardware Installation Manual* for information on the HP 98629A interface card.

The Connection to the SRM Controller

Your workstation's HP 98629A card connects to an HP 98028A multiplexer by a cable. Other workstations may connect to the same multiplexer, with each workstation having a different node address.

The multiplexer has a cable connecting it to the SRM system controller. The controller end of the cable attaches to another HP 98629A interface card in the controller.

The Controller's Identification

The controller card's node address setting is normally 0 for all HP 98629A cards in the system's first SRM controller, 1 for the second controller, and so on.

The SRM system controller may contain several HP 98629A interface cards. Unique select code settings distinguish one HP 98629A interface in the controller from another.

Identification of Shared Disc(s)

Each SRM controller can have multiple shared disc drives connected to it. From your workstation's point of view (as shipped), only one shared disc is available. The Pascal operating system automatically configures to have Pascal volume #5: assigned to the primary (first) SRM system disc.

Additional discs on the SRM system are recognized only if your Pascal CTABLE program is modified. (The procedure is discussed in the chapter on "System Startup" and is typically done by your SRM system manager when the disc is configured to the SRM system.)

Unit Numbers for Shared Disc(s)
A workstation connected to an SRM normally has units #5: and #45: set up for SRM shared disc access. The use of two units is in keeping with the idea that there are usually two special volumes (the system volume and the default volume) through which most file accesses occur.

If the workstation is booted from SRM, unit #45: will automatically be configured to be the system volume and unit number #5: will be available for use as the default volume. If there is local mass storage, the system volume can be any volume you desire. To set the system and default volumes, use the What command from the Main Command Prompt.

Here is how the Filer's Volumes display might look right after booting up a workstation connected to the SRM and having no local mass storage:

```
Volumes on-line:
  1    CONSOLE:
  2    SYSTERM:
  5 #  SRM:
  6    PRINTER:
 45 *  SYSTEM42:
Prefix is - SRM:
```

You can see that the system starts out with #5: as the default volume (`Prefix is`) and #45: as the system volume.

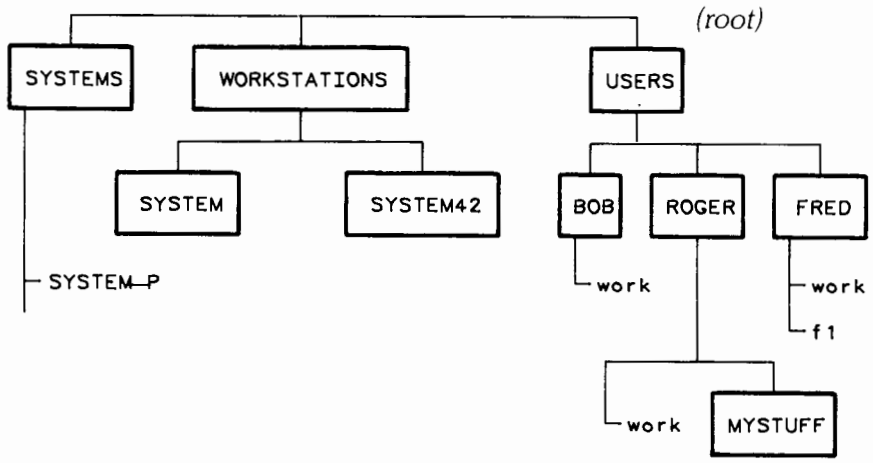
The names `SRM:` and `SYSTEM42:` are names of directories in the SRM's hierarchical directory structure (discussed later in this section). In this example, the name of the SRM volume is `SRM`, the workstation we are using is at node address 42, and a directory called *SYSTEM42* exists. `SYSTEM42:` is thus selected as the system volume (denoted by the * beside the directory's name). All of this selecting is done by the Pascal TABLE program as it configures the system each time you boot.

Identification of Shared Peripherals
Shared peripherals (printers and plotters) are connected to the SRM system controller through the controller's internal HP-IB interface. To manage the use of the shared printers and plotters, the controller maintains special directories called "spooler directories."

You send information, in files, to a peripheral by placing the file name in the proper spooler directory. The SRM controller keeps track of the order in which files are sent to the directories and sends the files to the appropriate output devices in that order. Once you have placed a file in a spooler directory, your workstation is free to do other processing.

SRM's Hierarchical Directory Structure

The SRM system uses a hierarchical directory structure to organize its files. This directory structure looks like an inverted tree (see example below). The top level in the structure is called the "root." Within the root are directories or files. Each directory may contain files or other directories. The drawing below shows a hierarchical directory structure.



The directory SYSTEMS is a special directory used by Boot ROMs version 3.0 or later to automatically load operating or language systems. The SYSTEMS directory is discussed in more detail in the "System Startup" chapter.

Note that, within the directory *USERS*, each of the three subordinate directories – *ROGER, BOB* and *FRED* – contains a file called *work*. Because each file and directory is uniquely specified by the list of directories from the root to the file, several files of the same name can exist in the directory structure (provided they are not within the same directory).

Notation

In specifying a location within the directory structure, the root is designated by a slash (/). To refer to a file or directory immediately under the root, for instance the directory *USERS* in the previous illustration, you would write:

/USERS

To specify a level further down the hierarchy, for instance, the directory *ROGER* under *USERS*, you would write:

/USERS/ROGER

and for yet another level:

/USERS/ROGER/work

As you can see, to specify a file, the list of directories to the target file must be specified. Directories in the list are separated by slashes (/). The sequence of names and slash delimiters is called a "directory path," because it indicates the path one must follow down the hierarchy to get to a particular file or directory.

SRM Access Rights

To control access to a file or directory, you can associate passwords with "attributes" that designate the type of access to be protected. Passwords may be assigned at the time the file or directory is created or with the Filer's Access command.

Any access right to which no password is assigned is public. Any user on the SRM may perform operations requiring the access right that is public, whereas only users supplying a correct password can perform operations that require the protected access.

The attributes and the access rights controlled by their association with passwords or made available to all SRM users by their remaining public are:

Attribute	Access to File	Access to Directory
READ	operations that read from the file	listing the directory's contents
WRITE	operations that write to the file	adding or modifying the directory's contents (also need SEARCH to add to directory's contents)
SEARCH	does not apply	accessing the directory's contents
CREATELINK	creating links to the file	creating links to the directory
PURGELINK	removing or renaming the file	removing or renaming the directory
MANAGER	assigning, removing or changing passwords on the file	assigning, removing or changing passwords on the directory
	when password-protected, gives all other access rights as well	when password-protected, gives all other access rights as well

For example, SEARCH access is required on all directories along the path to a given file or directory. Suppose the password *search_pass* protects the SEARCH access on the directory *ROGER* (see illustration). To reach the directory *MYSTUFF* from the root, a user would have to include the password in the directory path specifier to *MYSTUFF*, as in:

/USERS/ROGER<search_pass>/MYSTUFF

If the SEARCH access capability on *ROGER* was public, any user could "pass through" *ROGER* without supplying a password.

SRM access information is shown in the Filer's Extended directory listing under the heading `directory info`. The one-letter identifiers (`R` for READ, `W` for WRITE, `S` for SEARCH, and so forth) indicate the public access rights on the files listed.

The *Pascal Workstation System* manual discusses SRM access rights in more detail.

SRM Concurrent File Access

The Pascal system defines three modes of access to shared files: EXCLUSIVE, SHARED and LOCKABLE.

EXCLUSIVE access mode allows only one workstation to open a file at a time. All files opened on the SRM are, by default, opened in EXCLUSIVE mode.

A file whose access mode is SHARED may be opened by a number of SRM workstations for both reading and writing. This mode is primarily intended for reading from files, not writing to them. When two users write to the same file at the same time, results are unpredictable.

The LOCKABLE access mode restricts the ability to read from and write to a file to the workstation that locks the file (using the file operations LOCK, WAITFORLOCK, and UNLOCK, as described in the *Pascal Workstation System* manual). The file remains inaccessible to other workstations until it is unlocked from the workstation at which it was locked.

Files are typically locked and unlocked from within programs. The "Filer System" chapter of the *Pascal Workstation System* manual discusses concurrent file access in more detail.

The current status for SRM files is shown in the Filer's Extended directory listing under the heading `directory info`. A current status of the files within the listed directory is shown as `CLOSED`, `SHARED`, `EXCLUSIVE` or `CORRUPT`.

`CORRUPT` indicates that the disc lost power while accessing the file, possibly altering the file's contents.

Note

If a file's status is shown as `CORRUPT`, you should run the DSCK Utility program to check the directory structure and its integrity on the SRM system disc. Refer to the *SRM Operating System Manual* for details.

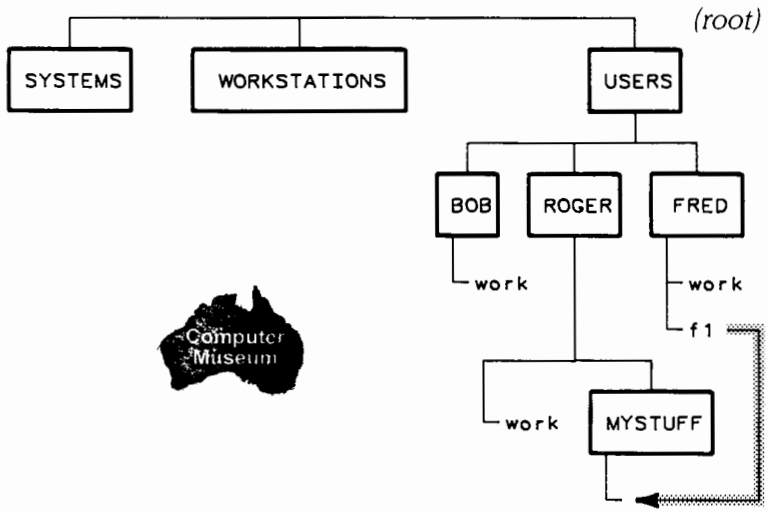
Duplicating Files in More Than One Directory

To save disc space, you can use the Filer's Duplicate link command to link a file into a directory other than its original location. Once a link has been established, the file looks to the Pascal system as though it is in the directory to which it is linked.

For example, if you created a duplicate link from the file *f1* in the directory *FRED* to the directory *MYSTUFF*, you could specify the path to *f1* as:

/USERS/ROGER/MYSTUFF/f1

even though no copy of *f1* actually exists in *MYSTUFF*.



This allows you to have access to files without making extra copies of them.

If a file is purged from a directory to which it was linked, only the directory from which the file was purged loses access to that file. All other directories with links to the file can still find it. The disc space allocated to a file is only reclaimed when no directories have links to that file.

The Duplicate link command is discussed with other Filer Commands in "The Filer" chapter of the *Pascal Workstation System* manual.

Using Your Pascal Workstation on SRM

This section describes, through examples, some common procedures you'll be using to operate your Pascal workstation on the SRM, including:

- specifying locations within the hierarchical directory structure;
- creating directories;
- specifying files, directories and volumes;
- protecting access to files and directories by assigning passwords;
- using shared printers and plotters;
- using Pascal Filer commands with SRM.

Note About Key References

Throughout this section, symbols for the keys used to enter commands are shown with each command.

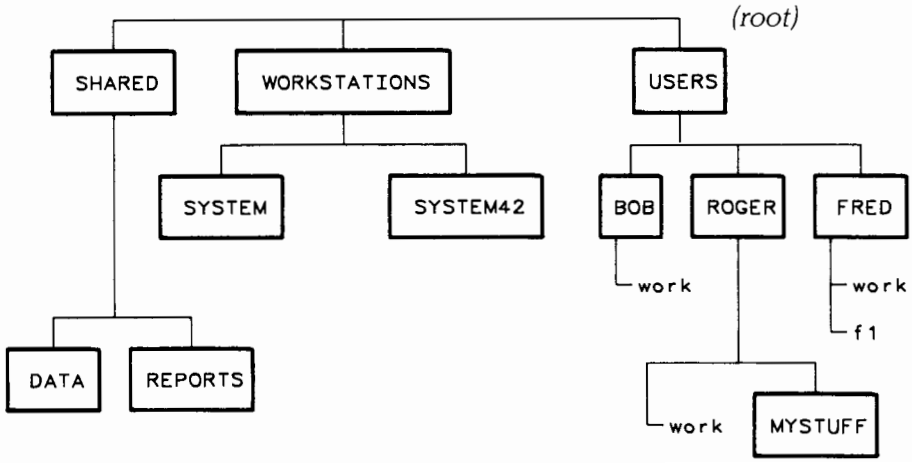
The [ENTER] symbol denotes the key to be used on either the HP 98203A or HP 98203B keyboard. The [Return] symbol denotes the key to be used on the HP 46020A keyboard.

The HP 46020A keyboard is not supported by versions of the Pascal language system prior to 3.0.

Moving Up and Down the Hierarchy

It would be tedious to type a directory path every time you wanted to access a file. To avoid this, you can use the Filer's Prefix and Unit directory (`Udir`) commands to establish a directory as a reference point from which another location in the directory structure can be specified. A directory path to a file begins either from that reference point or from the root. To move about within the directory structure, you specify new reference points.

The following discussion uses the directory structure illustrated below:



Using the Prefix Command

Specifying a default volume tells the Pascal file system what volume name to use when none is specified with a file name. With SRM, the Pascal file system considers a directory a "volume." Thus, you may designate a directory as the default volume and the system will perform file accesses directly within that directory or use the directory as the beginning of a path to another location in the directory structure.

All volumes recognized by your Pascal workstation are listed when you use the Filer's Volumes command. An example Volumes display for the directory structure illustrated above is:

```
Volumes on-line:
  1   CONSOLE:
  2   SYSTERM:
  5 # ROGER:
  6   PRINTER:
 45 * SYSTEM42:
 46 # SHARED:
Prefix is - ROGER:
```

In this Volumes listing, the directory *ROGER* is the current default volume. You may designate as the default volume any of the names that are shown in a Volumes listing with the # or * symbol beside them. Because these "volumes" are SRM directory names, you may also designate as the default volume any directory whose location can be specified through one of the names shown.

For example, to designate the directory *MYSTUFF* as the default volume, you would:

1. Type (P)

   The Filer prompts:

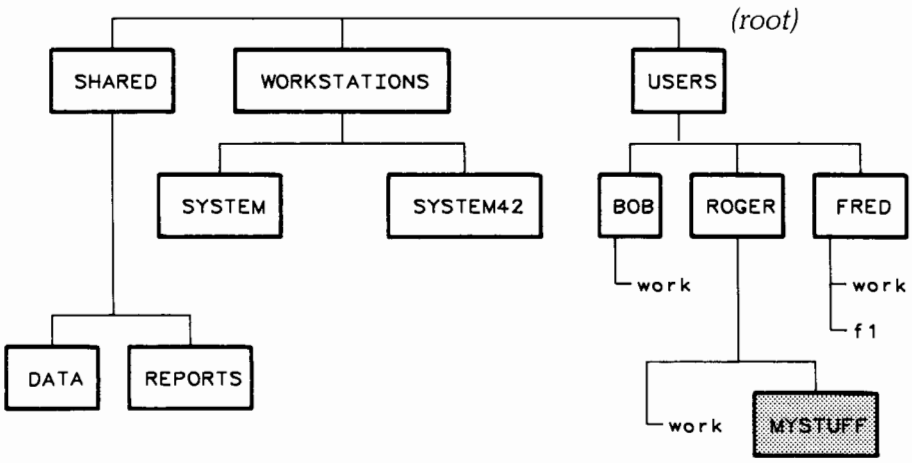
   ```
   Prefix to what directory?
   ```

2. Type

   MYSTUFF (ENTER) or (Return)

With *ROGER:* as the current default volume, the system would assume the path began at *ROGER.* The new Volumes listing (illustrated below) would show *MYSTUFF:* as the default volume and unit #5 would now be associated with *MYSTUFF:*

```
Volumes on-line:
  1   CONSOLE:
  2   SYSTERM:
  5 # MYSTUFF:
  6   PRINTER:
 45 * SYSTEM42:
 46 # SHARED:
Prefix is - MYSTUFF:
```



If you then wanted to list the contents of the directory *MYSTUFF*, you would:

1. Type (L) for the Filer's List directory command. The Filer prompts:

   `List what directory?`

2. Type

   : (ENTER) or (Return) *(notation that means "default volume")*

Next, if you used the Prefix command, answering the `Prefix to what directory?` prompt with:

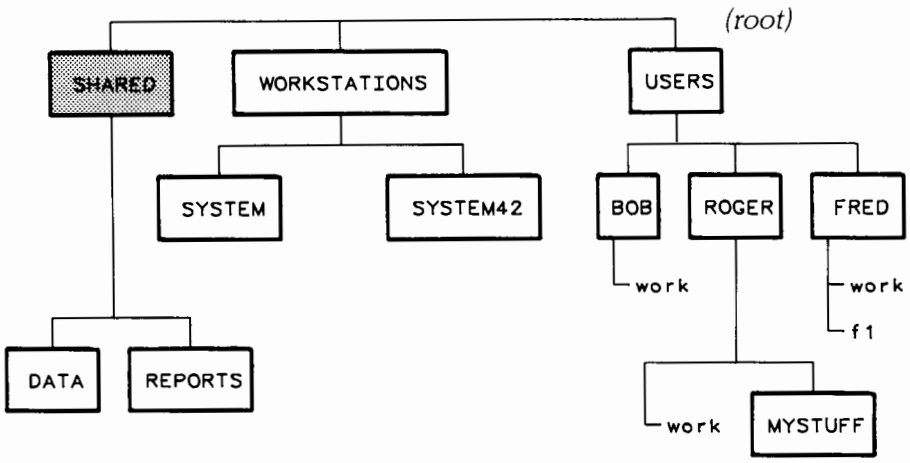
`#46:` (ENTER) or (Return)

or

`SHARED:` (ENTER) or (Return)

your default volume would be at unit #46, not at unit #5.

```
Volumes on-line:
  1   CONSOLE:
  2   SYSTERM:
  5 # ROGER:
  6   PRINTER:
 45 * SYSTEM42:
 46 # SHARED:
Prefix is - SHARED:
```



If you then used the Prefix command and responded to the prompt with:

`#5:/USERS/BOB` (ENTER) or (Return)

you would not only set the default volume to *BOB:* but you would also change the volume association for unit #5 from *ROGER:* to *BOB:*.

```
Volumes on-line:
  1   CONSOLE:
  2   SYSTERM:
  5 * BOB:
  6   PRINTER:
45 * SYSTEM42:
46 # SHARED:
Prefix is - BOB:
```

The Unit Directory Command

Another way to change the unit number/volume name association is with the Filer's Unit directory (`Udir`) command. This association is referred to as setting a "working directory." For example, typing:

(U) (*The Filer prompts* `Set unit to what directory?`)

`#46:DATA` (ENTER) or (Return)

would make *DATA* a working directory by associating unit #46 with that directory, but would not affect the default volume designation (`Prefix` is still *BOB:*).

```
Volumes on-line:
  1   CONSOLE:
  2   SYSTERM:
  5 # BOB:
  6   PRINTER:
 45 * SYSTEM42:
 46 # DATA:
Prefix is - BOB:
```

Now, because unit #46 is identified with the directory *DATA*, you would be able to use `#46:` and `DATA:` interchangeably with Filer commands. For example, to list the contents of *DATA*, you could respond to the List directory command's `List what directory?` prompt with either:

`#46:` (ENTER) or (Return)

or

`DATA:` (ENTER) or (Return)

Take care when using the Unit directory command **not** to assign the default volume's associated unit number to a name other than that of the current default volume. For example, if you were to type:

(U) (*The Filer prompts* `Set unit to what directory?`)

```
#5:/USERS/ROGER
```

the resulting Volumes display would show:

```
Volumes on-line:
  1   CONSOLE:
  2   SYSTERM:
  5 # ROGER:
  6   PRINTER:
 45 * SYSTEM42:
 46 # DATA:
Prefix is - BOB:
```

Even though *BOB:* would still be listed as the default volume, you would not be able use *BOB:* as the default volume, because *BOB:* is no longer an "on-line" volume. For instance, to list the contents of the default volume (directory), you would normally:

1. Type (L) (*The Filer prompts* `List what directory?`)
2. Type

   `:` (*which is the notation indicating the default volume*) (ENTER) or (Return)

   But because, in this example, the Filer looks for a volume named `BOB:` that is associated with one of the unit numbers and doesn't find it, it does not list the directory's contents, as requested.

Note

You should **not** use the Unit directory command to change the association of #45: from your SYSTEM*nn* directory (the system volume). Doing so causes the system to lose access to the system files because the Pascal system looks for a workstation's system files in its SYSTEM*nn* directory.

You may, however, assign any of the on-line volumes as the default volume without affecting the system's access to the system files.

Moving Up the Hierarchical Directory Structure

A special directory name is provided for moving up the hierarchy. Two periods (..) can be used to denote the directory containing the current directory. For instance, if *MYSTUFF:* were the default volume, you could make *ROGER:* the default volume by answering the Prefix command's `Prefix to what directory?` prompt with:

`..` ENTER or Return

To go up two levels, use the double-period twice, separated by a slash. For example to change the default volume from *MYSTUFF:* to *USERS:*, you would respond to the Prefix command's prompt with:

`../..` ENTER or Return

The " .. " notation can be used to move all the way up the directory structure to the root, although, if you want to go directly to the root, using the slash (/) root identifier is easier.

Creating Directories

SRM directories are created by the Filer's Make command. For example, to create a directory called *KAREN* in *USERS* you would:

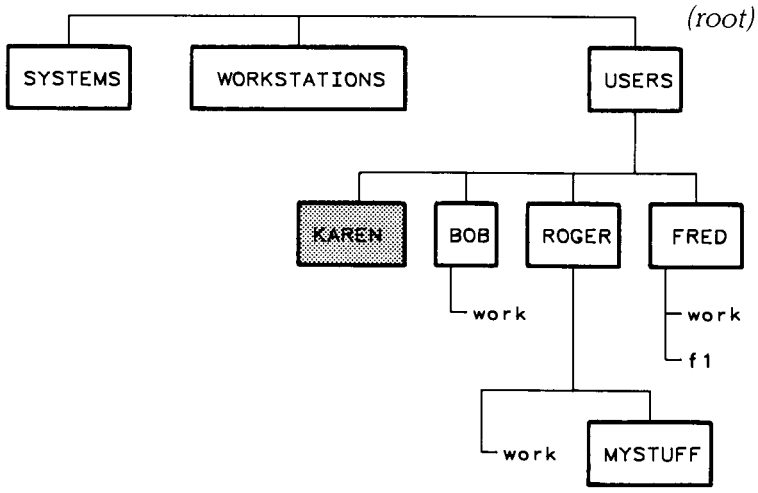
1. Type M

   The Filer prompts:

   ```
   Make File or Directory ? (F/D)
   ```

2. Type D and specify where you want the directory located:

   `#5:/USERS/KAREN` ENTER or Return



Specifying Files, Directories and Volumes

Syntax of File or Directory Specification

The syntax of a legal *file_specification* is given by:

file_specification:: = [volume_id] [directory_path] file_name ["["size_spec"]"]

:: = volume_id

In this notation, items between " [" and "] " are optional and quoted items appear literally. Note that, because a directory is a type of file, **specification for directories is the same as that for files.**

A *file_specification* may appear in one of two forms. The first form consists of an optional *volume_id* followed by a colon (:), then an optional *directory_path*, then a *file_name* which is not optional, then an optional *size_spec*.

An example of the first form is:

```
#45:SYSTEMS/FILER
```

The second form consists of a *volume_id* only. An example of the second form is:

```
#5:
```

Syntax of a Volume Identifier

The volume identifier (*volume_id*) selects one of up to 50 logical units known to the Pascal file system. If no *volume_id* is present, the volume used is the default volume (selected by the Filer's Prefix command). Otherwise, the volume is specified in one of three ways:

volume_id:: = "#"integer[password]":"

:: = ":"

:: = name[password]":"

In the first case, *integer* is a two-digit number from one to 50. For example, `#23:` is a valid volume identifier. The second case is a special form denoting the default volume. In the third case, *name* is a sequence of characters.

If the volume *name* of the SRM disc is *DISC_ONE*, the disc could be specified as:

```
DISC_ONE:
```

For a logical unit connected to an SRM system, the *volume_id* takes a special meaning. The notation `#5:` refers to the working directory of unit number five. The notation `#5:/` refers to the root of the SRM directory structure, with which unit number five is associated. The working directory for any SRM volume is selected by the Filer's Prefix or Unit directory commands (refer to the "Moving Up and Down the Hierarchy" section earlier in this chapter), or the What command of the Main Command Prompt.

Passwords

Passwords are sequences of up to 16 characters that govern the access rights to a file, a directory or the SRM volume (shared disc). Passwords are assigned to a file or directory either when it is created or by the Filer's Access command. The SRM volume password is assigned through the SRM operating system's RENAME or INITIALIZE commands (refer to the *SRM Operating System Manual* for details).

Including the SRM disc's volume password in a *file_specification* gives you unlimited access to all directories and files on the shared disc. The volume password overrides all other passwords in the system. Because of its power, this password is usually protected and used only with proper authority.

You may use either of the forms for the volume password illustrated in the examples below:

```
#5<volpassword>:/USERS/ROGER/work
#5:<volpassword>/USERS/ROGER/work
```

That is, the volume password may either immediately precede or follow the colon separator.

Syntax of a Directory Path

Directory paths are allowed in Pascal *file_specification* only when specifying files on SRM logical units. The syntax for a *directory_path* is:

directory_path:: = ["/"] {directory_name [password]"/"}

password:: = "<" word ">"

directory_name:: = file_name

:: = "."

:: = ".."

The information between " { " and " } " may occur zero or more times. As you can see, there are two special directory names allowed with the SRM. The name " . " (a single period) refers to the current working directory. The name " .. " refers to the directory containing the working directory. Other names in a *directory_path* are directories along the path to the file or directory being specified.

Note that a *directory_path* doesn't appear by itself. It appears as part of a *file_specification*, with the *file_name* following the *directory path*. Examples of directory paths are:

```
/.<PASS1>/          (denotes root, using password PASS1)
/USERS/ROGER/       (denotes directory ROGER in USERS, where USERS is at the root)
```

A specifier including both the *directory_path* and *volume_id* might appear as follows:

```
#5:/WORKSTATIONS/SYSTEM13
```

SRM File or Directory Names

The SRM system allows almost any file name. The Pascal system removes blanks and control characters from file names.

The Pascal SRM Directory Access Method interprets the " < " character as the beginning of a password. All characters up to the next " > " character constitute the password.

File Size Specification

The last, optional part of a *file_specification* is the file size specifier (*size_spec*). Its syntax is:

size_spec:: = "[" integer "]"

:: = "[*]"

This specification takes effect only if a new file is being created with REWRITE, OPEN, APPEND or APPEND with OPEN. If the file already exists, the file system tries to make the file at least the size specified. The size is ignored for RESET.

In the first form, the integer gives the number of 512-byte blocks to be allocated to the file. For instance `[100]` would cause allocation of 51,200 bytes.

The second form `[*]` specifies that the file is to be allocated either half of the largest free space or the second largest free space, whichever is larger. If no size is specified when space for a new file is being allocated, the largest free area is assigned to the file.

SRM extends the space allocation for files as needed. If data added to a file would cause the file to overflow its original space allocation, the system adds another block of memory (contiguous, if possible) called an "extent" to the file's current space allocation. An extent's size is equal to the original space allocation for the file.

Allowable File or Directory Names

What file names are allowed depends on the type of directory used on the volume in which the file resides. In other words, the directory organization determines the file name rules.

File or directory names can consist of alphabetic letters and digits and the the hyphen (-), underscore (_), and period (.) characters. Blanks are removed from file names.

In SRM directories, uppercase and lowercase letters are distinct. (ROGER is not the same as Roger.) You should not use the " / ", " < " and " > " characters, which have special meaning to the SRM system, in file or directory names.

Protecting Access to Files and Directories

The Filer's Access command allows you to change public access rights on your SRM files and directories.

To use the Access command,

1. Type (A). The Filer prompts:

   ```
   Access rights for what file?
   ```

2. Type the file or directory specification. For example:

   ```
   #5:/USERS/ROGER/work
   ```

   If the file's MANAGER access is password-protected, you must include the password in the file/directory specification.

   The Filer then prompts:

   ```
   Access: List, Make, Remove, Attributes, Quit?
   ```

3. You may then `List` the existing password(s) and the attributes to which each is assigned, `Make` new password/attribute assignments or `Remove` passwords. `Attributes` is a help feature that lists the attributes options. `Quit` returns you to the Filer prompt.

The attribute option ALL is a shortcut notation for assigning a password to protect all access rights to a file or directory. For example, to assign the password *all_rights* to protect all access rights on the file *work* in the directory *ROGER* (assume all access rights on the directories in the path to *work* and on *work* itself are public), you would:

a. Type (M). The Filer prompts:

   ```
   Make password:attribute?
   ```

b. Type:

   ```
   all_rights:ALL
   ```
   (ENTER) or (Return)

Now, any user wishing to access *work* for operations requiring any of the access rights (READ, WRITE, MANAGER, SEARCH, SEARCHLINK, PURGELINK) must include the password in the file specifier, for example:

```
#5:/USERS/ROGER/work<all_rights>
```

For more information, refer to the description of the Access command in "The Filer" chapter of the *Pascal Workstation System* manual (previously the *Pascal User's Manual*).

Using Shared Printers and Plotters

Using a shared printer or plotter to output data requires you to place your data in a file in the spooler directory. Once the file is in the directory, the SRM operating system sends the file to the appropriate output device as soon as the device is free.

For example, to print the text file named *JOB_1.TEXT* located by the SRM directory path *#5:/PROJECT_1* on the printer assigned to spooler directory named *LP*:

1. Type F to enter the Filer from the Main Command prompt.
2. Type T to enter the Filer's Translate mode.

   The Filer prompts:

   ```
   Translate what file?
   ```

3. Type:

   ```
   #5:/PROJECT_1/JOB_1.TEXT,#5:/LP/JOB_1.ASC
   ```

   ENTER or Return

The file will be printed as soon as the printer is available. The `.ASC` ending on the file name tells the Filer to translate the information file into ASCII format, which is best handled by the SRM and its supported peripherals.

In contrast to the SRM 1.0 system, the SRM 2.0 operating system allows non-ASCII files to be sent to the printing device as a byte stream.

Note

Any non-ASCII (such as .TEXT) file sent to the spooler is printed exactly as the byte stream sent. Unless you set up your non-ASCII file correctly, improper printer output or operation could result. Therefore, it is recommended that you use only ASCII type files when spooling to a printer or plotter.

Using Pascal Filer Commands With SRM

The following table summarizes the Filer commands either used exclusively with SRM or whose use has a special meaning for SRM. For full syntax and semantics information about these commands, refer to "The Filer" chapter of the *Pascal Workstation System* manual (previously the *Pascal User's Manual*).

Command	Significance to SRM
`Access`	Changes the access rights (passwords) on a file or directory
`Bad_secs`	Not valid for SRM
`Duplicate`	Links a file or files to a directory at another location in the directory structure
`Krunch`	Not valid for SRM
`Make`	Creates a directory within the SRM directory structure
`Prefix`	Sets or changes the default volume name
`Udir`	Changes the path specification for a unit directory
`Zero`	Not valid for SRM

Chapter 3 System Startup

This chapter describes the procedures required to bring up an HP Series 200 workstation on the SRM for the first time. The procedures described in this chapter supplement the installation procedures described in the "System Installation" chapter of the *SRM Operating System Manual* and are to be performed by a system manager.

The procedures described in this chapter are required for bringing up only the **first** of the language system's workstations on the SRM for the **first** time and do not apply to adding subsequent workstations to the SRM system.

This chapter is divided into two parts that describe the startup procedures for use from a BASIC SRM workstation and from a Pascal SRM workstation. Each part tells how to install the system software required to access and use the SRM with the language system and discusses the SRM directory structures recommended for use with workstations operating under that language system.

Initial System Startup From a BASIC SRM Workstation

The following instructions describe the procedure for bringing up the SRM system on an HP Series 200 BASIC 3.0 workstation for the first time. The instructions assume that the workstation has its own floppy disc drive.

The "Initial System Startup From a Pascal SRM Workstation" section in this chapter gives instructions for bringing up the SRM system on an HP Series 200 Pascal workstation.

This section describes system setup procedures you perform from a BASIC workstation, including:

- Creating the necessary directories on the system disc;
- Copying operating system files to the system disc;
- Creating files for automatic configuration at bootup (optional).

Note

This section reflects the startup procedures valid for use with BASIC 3.0. The "SRM and BASIC 2.0" appendix gives the startup procedures appropriate for BASIC 2.0.

Creating Directories on the System Disc

This procedure follows initial installation of the SRM operating system on the system controller. You must have performed steps one through eight in the first section of the "System Installation" chapter, *SRM Operating System Manual*, before performing this procedure.

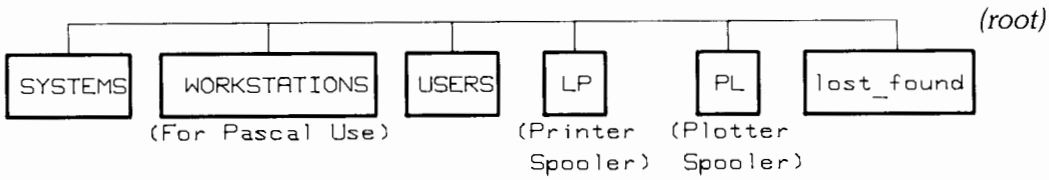
The following steps load the BIN files necessary for accessing the SRM from an HP Series 200 BASIC workstation and create the SYSTEMS and USERS directories on the SRM system disc. Use of the two directories is recommended to reduce the number of directories and files at the root level of the SRM directory structure.

Note About Key References

Throughout this section, symbols for the keys used to execute statements and commands are shown with each statement or command.

The [EXECUTE] symbol denotes the execution key on either the HP 98203A or HP 98203B keyboards (the key on the HP 98203A keyboard is labeled [EXEC]). The [Return] symbol denotes the execution key on the HP 46020A keyboard.

You may also use the [ENTER] key on these keyboards to execute commands.



Recommended Root-Level Directories

NOTE: For a description of the WORKSTATIONS directory, refer to the section on "Initial System Startup From a Pascal SRM Workstation" later in this chapter.

You create the SYSTEMS directory to hold operating system and bootup configuration files. The procedures for placing files in the SYSTEMS directory and for creating bootup configuration files are described in the next section.

The USERS directory allows you to group working directories for the individual users (workstations) on the SRM system.

1. With your computer's power OFF, insert the BASIC 3.0 SYSTEM DISC into the workstation's primary drive. Refer to the operating manual for the HP Series 200 computer or for the disc drive to determine which is the primary drive.

   Turn the computer's power ON and allow the BASIC operating system to completely load, then remove the disc.

2. Insert the BASIC 3.0 LANGUAGE EXTENSIONS DISC into the primary drive and type:

   ```
   LOAD BIN "SRM"
   ```
   EXECUTE or Return

3. After the SRM BIN file is loaded, remove the LANGUAGE EXTENSIONS DISC from the primary drive, insert the BASIC 3.0 DRIVERS DISC into the same drive, and type:

   ```
   LOAD BIN "DCOMM"
   ```
   EXECUTE or Return

4. To establish communications with the SRM, type:

   ```
   MSI ":REMOTE"
   ```
   EXECUTE or Return

   which places you at the root of the SRM directory structure. You may wish to verify your location within the directory structure by typing:

   ```
   CAT
   ```
   EXECUTE or Return

   The header for the resulting catalog listing should indicate the remote msus as shown in the example below:

   ```
   :REMOTE 21, 0                    remote msus
   LABEL:     SRM
   FORMAT:    SDF
   AVAILABLE SPACE:        54096
   ```

5. Create the SYSTEMS directory by typing:

   ```
   CREATE DIR "SYSTEMS"
   ```
   EXECUTE or Return

6. Create the USERS directory by typing:

   ```
   CREATE DIR "USERS"
   ```
   (EXECUTE) or (Return)

7. Leaving the workstation connected to the SRM under the BASIC operating system, return to Step 9 in the "System Installation" chapter of the *SRM Operating System Manual*, and proceed with installing the SRM operating system file, SYSTEM_SRM, in the SYSTEMS directory.

The following discussion and examples assume that SYSTEM_SRM is the first system installed on the system disc and in the SYSTEMS directory. Besides installing the SRM operating system on the system disc, you may wish to:

- create the spooler directories;
- copy other operating system files into the SYSTEMS directory;
- create workstation bootup configuration files (optional);
- create individual working directories within the USERS directory.

Setting up spooler directories is discussed in the "System Installation" chapter of the *SRM Operating System Manual*. Use of the SAVE command to save the spooler configuration and node names creates and updates the file, CONFIG_SRM in the SYSTEMS directory (also discussed in the "System Installation" chapter of the *SRM Operating System Manual*).

The next sections discuss how to design and implement a bootup scheme for workstations on your SRM system.

Planning the SYSTEMS Directory

Creating the SYSTEMS directory allows you to concentrate all operating system software and configuration files in a single directory on the SRM system disc.

You will determine your SYSTEMS directory's contents and structure according to the SRM system's application within your environment. Your decision depends upon the operating system software your users require and upon the powerup scheme you wish to implement for the workstations on your network.

In general, environments supporting BASIC workstations fall into one of two categories:

- those in which all or most of the users operate primarily under the BASIC operating system;
- those supporting users of several operating systems.

In the first environment, you may want all workstations to boot the BASIC operating system at powerup, whereas in the second, you may want each workstation to boot the operating system of its user's choice.

Note

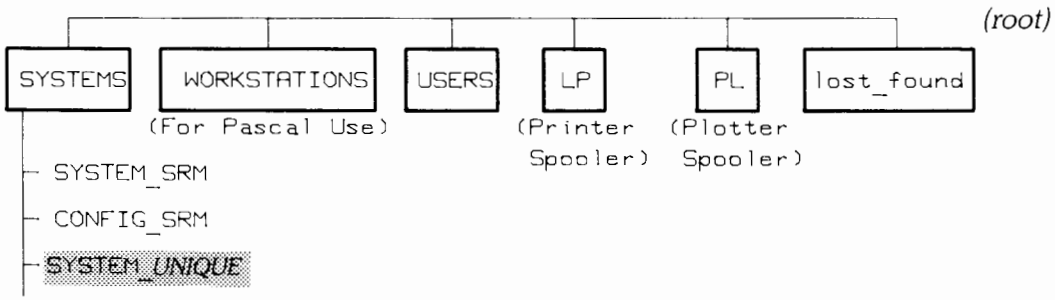
Only HP Series 200 computers with Boot ROM version 3.0 or later can boot from the SRM. Upon powerup, the computer should display `BOOTROM 3.0`. If no boot ROM message is displayed, the computer has an earlier version boot ROM. If `BOOTROM 3.0L` is displayed, the computer's boot ROM is a subset of the 3.0 boot ROM and does not support automatic booting from SRM.

The following sections help you plan your SYSTEMS directory to accommodate the workstations capable of booting from the SRM. Workstations with ROMs that do not support automatic booting must boot from local mass storage.

If BASIC is the Predominant Operating System

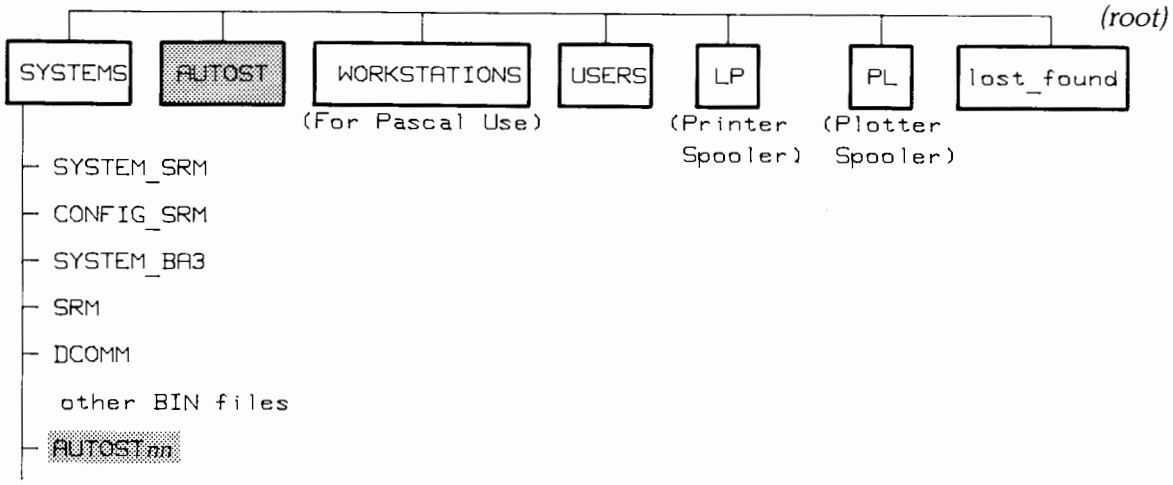
The order of the files in the SYSTEMS directory determines, to some extent, the operating system software used by the workstation boot ROM at powerup. Unless the workstation operator interrupts the bootup procedure to select a system, the boot ROM loads the first bootable system file (a file of type SYSTM whose name begins with `SYS`) from the SYSTEMS directory into the workstation.

If all users on your SRM system require the same basic set of BIN files, you may wish to create a unique BASIC system file for use at powerup. Refer to the *BASIC Language Reference* manual for details on creating this file (see the STORE SYSTEM keyword). Your SYSTEMS directory would look similar to the one illustrated below:



where *UNIQUE* is the name you assign your customized BASIC system file. Note that you must include the SRM and DCOMM BIN files in that file for the workstations to communicate with the SRM system. Refer to the section on "Placing System Files in the SYSTEMS Directory" for more information on which BIN files to include.

If each user requires a unique set of BIN files, you may prefer to keep separate BIN files in the SYSTEMS directory and create autostart files for individual workstations. Your SYSTEMS directory would look similar to the list below:



where each AUTOST*nn* is an autostart PROG file specifying the initial BASIC system operations to be performed for a specific workstation on the network. The AUTOST file specifies the default set of operations, used for workstations for which no unique autostart file exists.

(Note that AUTOST is at the root level of the SRM directory structure, not within SYSTEMS.) Refer to the section on "Creating Autostart Files for Use by the BASIC 3.0 System" for further details.

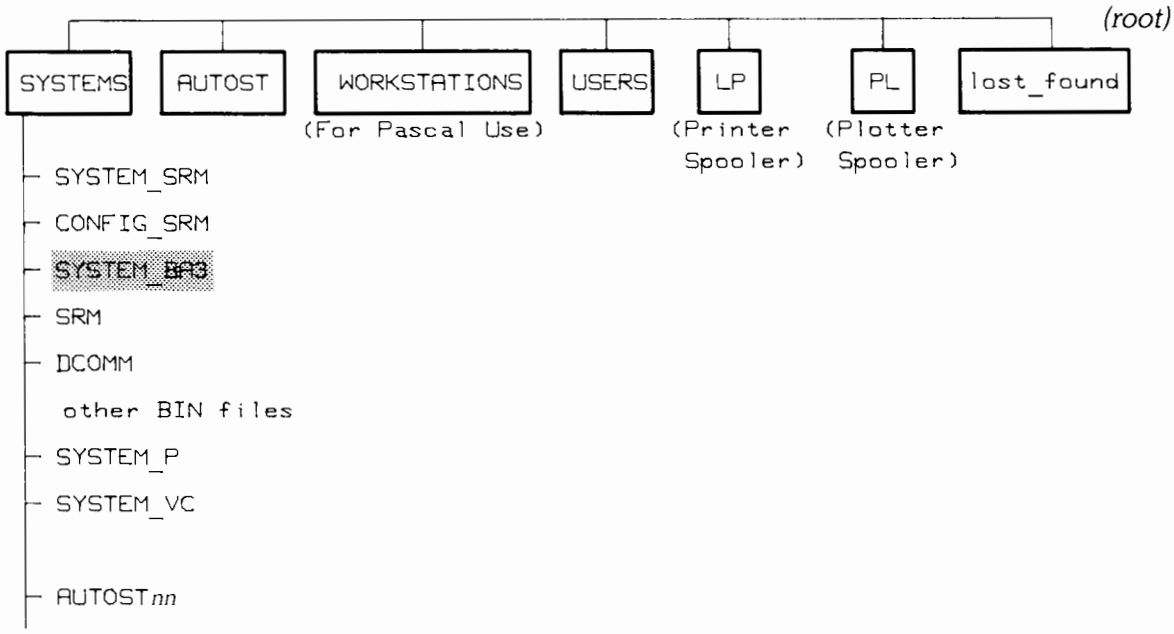
If a Variety of Operating Systems are Used

If you want each workstation on the network to power up in the operating system of its user's choice, you may construct the SYSTEMS directory in either of two ways.

A workstation user can always select an operating system from among those available to the workstation. If a user presses any key immediately after the powerup display indicates that the boot ROM has recognized the keyboard, the ROM lists all available bootable systems and waits for the user to select a system. Refer to the *BASIC User's Guide* for further details on explicit system selection at powerup.

If you want to give users a choice of systems by this method, the order of the systems files in your SYSTEMS directory is unimportant, except that the boot ROM will use the first bootable system file listed in the directory if the user does not interrupt booting to select a system. A bootable system file is a file of type SYSTM whose name begins with SYS.

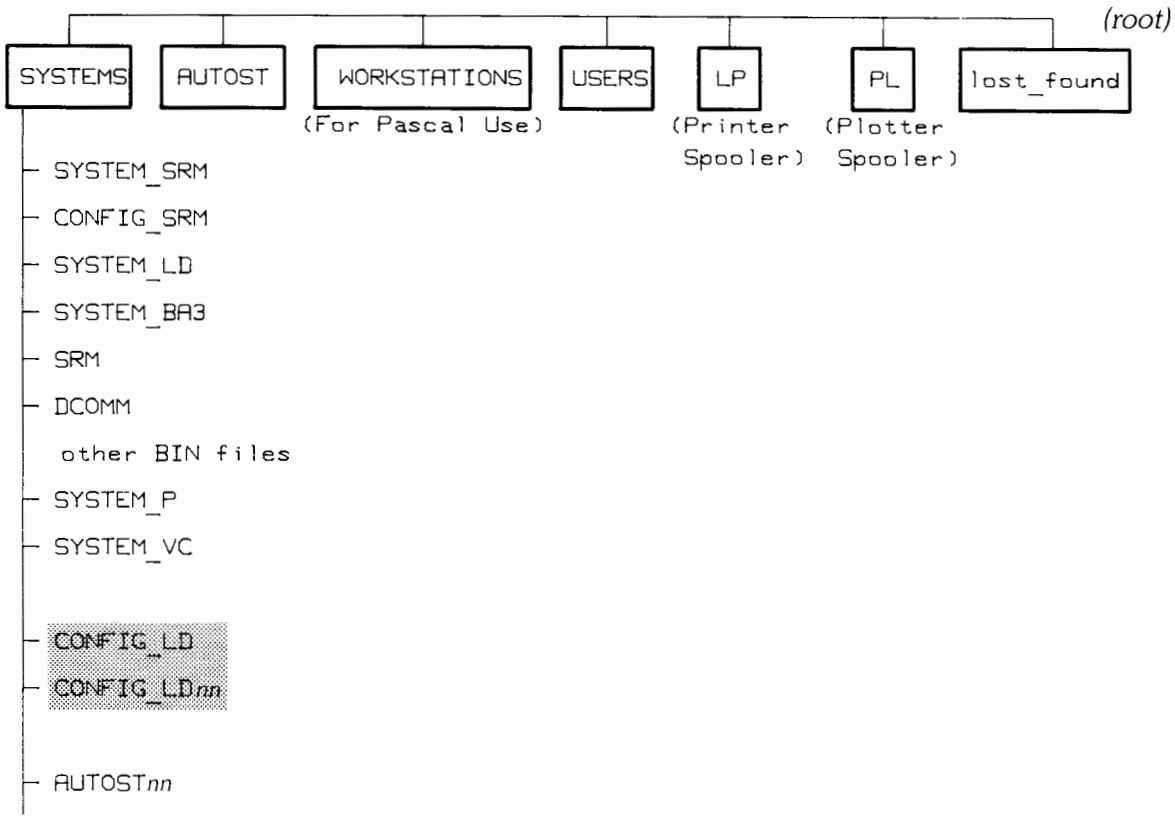
For example, if the files in your SYSTEMS directory were ordered as listed as below:



if a workstation user on the SRM system did not specifically select an operating system at powerup, the boot ROM would boot SYSTEM_BA3 in the workstation, along with BIN files specified in the appropriate AUTOST or AUTOST*nn* file. Users could select another system, such as Pascal (SYSTEM_P) instead by interrupting the booting process as described above.

To automate the process, you may use the BASIC Loader Utility (also called the "secondary loader"), which selects the operating system for the user. Use of the Loader Utility requires you to place the BASIC Loader Utility system file (SYSTEM_LD) in SYSTEMS as the first bootable system file. The Loader Utility requires special configuration files specifying the system to be selected, so you must also include those files in the SYSTEMS directory.

The next sections explain the Loader Utility method in more detail. If you use this method, your SYSTEMS directory should look similar to the list below:



where each CONFIG_LD*nn* represents a configuration file specifying the system to be booted in a specific workstation on the network, and CONFIG_LD is the configuration file specifying the system to be booted in workstations for which no unique file exists. Refer to the section on "Creating Configuration Files for Use by the Loader Utility" for further details.

If autostart (AUTOST or AUTOST*nn*) file(s) are in the directory also (as in the illustration), and the CONFIG_LD file specifies the BASIC 3.0 system, upon being loaded, the BASIC system automatically performs the procedures specified in the autostart files.

Note that, with this method, the order of all files in the SYSTEMS directory, other than SYSTEM_LD, is unimportant.

Placing System Files in the SYSTEMS Directory

You may now place files in the SYSTEMS directory in the order required for your chosen workstation bootup scheme (as discussed in the previous section). This section lists the system files required for BASIC use on SRM, and illustrates the procedure for copying individual files from the BASIC discs to the SYSTEMS directory.

System Files Important for BASIC Use on SRM

The table below shows some of the operating system files you may wish to copy to the system disc, the importance of each file to SRM use, and the disc containing the file:

System File	Necessary if you wish to use:	On the disc labeled:
DCOMM BIN file	Shared Resources (SRM)	DRIVERS
GRAPH BIN file	PLOTTER IS <remote file> SYSTEM$("PLOTTER IS")	LANGUAGE EXTENSIONS
I/O BIN file	PRINTER IS (with EOL, WIDTH parameters) RESET	LANGUAGE EXTENSIONS
KBD BIN file	LOAD KEY STORE KEY RE-STORE KEY	LANGUAGE EXTENSIONS
MS BIN file	CAT (with options)	LANGUAGE EXTENSIONS
PDEV BIN file	LOADSUB FROM	LANGUAGE EXTENSIONS
SRM BIN file	Shared Resources (SRM)	LANGUAGE EXTENSIONS
TRANS BIN file	TRANSFER	LANGUAGE EXTENSIONS
SYSTEM_BA3	BASIC Operating System	BASIC 3.0 SYSTEM
SYSTEM_LD	BASIC Loader Utility	UTILITIES LIBRARY 2

Example: Placing the Loader Utility System File in the SYSTEMS Directory

To copy system files to the SYSTEMS directory on the SRM system disc, follow the procedure discussed below, substituting the name of the appropriate system file in the COPY command, and using the discs listed in the table above.

You should copy the files in the order in which they are to be placed in the SYSTEMS directory. This order is determined by your chosen workstation powerup scheme, as discussed in the section on "Planning the SYSTEMS Directory."

Working from the BASIC workstation you used to create the SYSTEMS directory, you would copy the BASIC Loader Utility program (shipped on the disc labeled UTILITIES LIBRARY 2) to the SYSTEMS directory by typing:

```
COPY "SYSTEM_LD:<msus>" TO "SYSTEMS/SYSTEM_LD" EXECUTE or Return
```

where *<msus>* is the mass storage unit specifier for the device from which you are copying SYSTEM_LD. If you are copying SYSTEM_LD from disc labeled UTILITIES LIBRARY 2, the *<msus>* specifies the disc drive containing the disc.

Refer to the "Data Storage and Retrieval" chapter of the *BASIC Programming Techniques* manual for details on *msus*.

Creating BASIC Workstation Bootup Configuration Files

This section describes how to create the files necessary to allow the BASIC workstations that are capable of booting from the SRM to boot from the SRM in a specific configuration. Two types of configuration files can be used for automatic configuration, either singly or in combination:

- Configuration file(s) for use by the Loader Utility. Using the CONFIG_LD files, the Loader Utility selects the operating system for the boot ROM to install in the workstation.
- Autostart file(s) for use by the BASIC 3.0 operating system. The AUTOST files specify some initial actions to be performed by the operating system, such as loading BIN files.

Creating Configuration Files for Use by the Loader Utility

If you want workstations on your SRM system to automatically boot selected systems, the BASIC Loader Utility system file, SYSTEM_LD should be the first bootable system file stored in the SYSTEMS directory. Use of the Loader Utility is not necessary if all workstations are to boot the same system at powerup.

For use with the SRM, you create two kinds of bootup configuration files -- a default file and a configuration file unique to a workstation. The Loader Utility uses the default configuration file for any workstation for which a unique configuration file does not exist.

The Loader Utility recognizes the default configuration file as CONFIG_LD, and individual workstation configuration files as CONFIG_LD*nn*, where *nn* is the workstation's assigned node number. For node numbers 0 through 9, *nn* is specified as a single digit. The Loader looks for its configuration files in the SYSTEMS directory.

Each configuration file is an ASCII file containing only the name of the operating system file to be loaded by the boot ROM at powerup. Any further operations, such as loading BIN files, must be specified in an autostart file.

The example program below creates the ASCII file *CONFIG_LD7* in the SYSTEMS directory to specify the BASIC 3.0 system as the bootup system for an SRM workstation at node number 07.

```
10  CREATE ASCII "/SYSTEMS/CONFIG_LD7",1
20  ASSIGN @Filename TO "/SYSTEMS/CONFIG_LD7"
30  OUTPUT @Filename; "SYSTEM_BA3"
40  ASSIGN @Filename TO *
50  END
```

For more details on creating the loader configuration files, refer to the BASIC *Loader Utility Manual.*

Creating Autostart Files for Use by the BASIC 3.0 System

Any workstation booting the BASIC 3.0 system may have an autostart file instructing the system to perform certain operations upon powerup. Autostart files are PROG files (whose names begin with `AUTOST`), which the BASIC system automatically loads and executes. An autostart file may include instructions to load BIN files (for example, `LOAD BIN "SRM"`).

For more details on creating autostart files, refer to the *BASIC User's Guide* or the "Entering, Running and Storing Programs" chapter of the *BASIC Programming Techniques* manual. For use with the SRM, you may create two kinds of autostart files -- a default file and an autostart file unique to a workstation. The BASIC system uses the default file for any workstation for which a unique autostart file does not exist.

When a workstation boots BASIC 3.0 from the SRM system disc, the BASIC system looks for a file named AUTOST*nn* in the SYSTEMS directory. *nn* is the node number assigned to the workstation, with node numbers 0 though 9 designated by a single digit (for example, *AUTOST6*). If a workstation's autostart file does not exist, the system then looks for the default file named AUTOST at the root level of the SRM directory structure. Failing to find either, the system waits for the first command from the keyboard.

Initial System Startup From a Pascal SRM Workstation

This section explains how to configure workstations to access and boot Pascal from an SRM system disc using three methods of modifying the standard configuration:

- Copying and re-naming files;
- Adding modules to INITLIB;
- Modifying the TABLE program (optional).

This section tells what to do the **first** time you set up the **first** Pascal workstation to access an SRM system. It should not be repeated for every workstation you set up. Once this procedure is complete, the SRM will be accessible to all Pascal workstations on the SRM system.

Note About Key References

Throughout this section, symbols for the keys used to enter commands are shown with each command.

The (ENTER) symbol denotes the key to be used on either the HP 98203A or HP 98203B keyboard. The (Return) symbol denotes the key to be used on the HP 46020A keyboard.

The HP 46020A keyboard is not supported by versions of the Pascal language system prior to 3.0.

Before You Begin

Before you begin the procedures outlined in this section, you should be aware of the following points:

Prerequisites

This procedure assumes:

- The person designated as the "SRM system manager" will perform the procedure.
- Your SRM hardware has been installed and tested as prescribed in the *SRM Hardware Installation Manual.*
- Each workstation in your SRM configuration has a unique node number. You will need to know the node numbers assigned to the workstations when your SRM system was designed and installed.

Boot ROM Versions

If you have an HP Model 216 Computer with Boot ROM 3.0L, you must boot from a local disc drive. The SRM can be used only after normal booting is complete. Similarly, if you have an HP Model 226 or 236 computer with a boot ROM whose version number is less than 3.0, you must boot from the internal 5.25-inch flexible disc drive. In either case, you may want to make a backup copy of the original BOOT: disc, as you will be modifying the INITLIB file on that disc.

Computers equipped with Boot ROM version 3.0 or later can boot directly from the SRM.

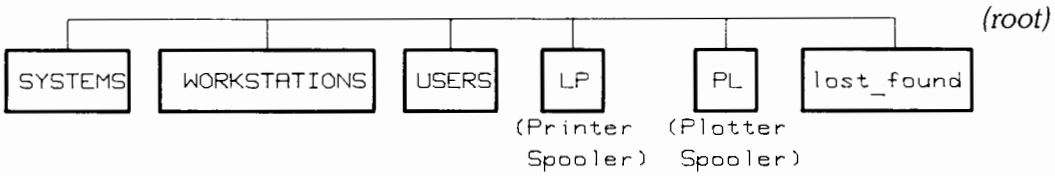
SRM Version 1.0 Operating System Parameters

Four parameters must be set when the SRM 1.0 operating system is initially configured. (With SRM 2.0, they are set automatically.) Appropriate values for these parameters when using Pascal workstations with SRM 1.0 are:

Parameter	Recommended Value
IOBUFFERS	At least 5 per workstation in the configuration (for example, 40 buffers for 8 workstations)
DISC BUFFERS	50
TASKS	2
FILES	10 or 12 open files per workstation in the configuration

Planning Your SRM Directory Structure

The first time you access an SRM system from a workstation, you need to set up certain directories on the SRM system disc. These directories have special functions, as described in the following paragraphs.

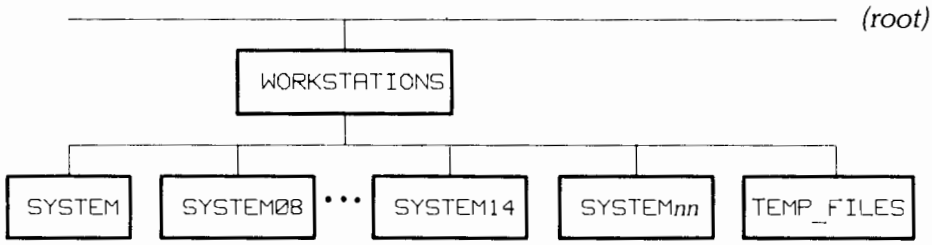


Recommended Root-Level Directories

This suggested directory structure calls for a directory named WORKSTATIONS in the root of the SRM directory structure, which is to contain several directories for Pascal workstation system files. These directories include:

- SYSTEM, which is to contain copies of all the Pascal system files, such as the Compiler, Filer and Editor.
- A SYSTEM*nn* directory for each workstation on the SRM system, where each *nn* corresponds to a workstation's node number.
- TEMP_FILES, within which all temporary files are kept.

To allow each workstation in an SRM configuration to boot a unique system and have its own system volume, you may wish to establish a unique SYSTEM directory for each node number.



Recommended Contents of the WORKSTATIONS Directory

For example, the illustration shows SYSTEM directories for workstations on nodes 08 and 14.

If a workstation has a local high-performance mass storage device, you may wish to use that device as the workstation's system volume. In fact, the automatic configuration process selects a high-performance mass storage (if one is present) as the system volume. The suggested directory structure uses separate directories for each node, with the option to copy frequently used files, such as the Editor and Compiler, from the SRM system disc onto local high-performance system volumes. Then, when a workstation boots the system, those files can be accessed locally with correspondingly greater speed.

Each workstation's SYSTEM directory should provide access to all the system files normally used by the workstation. For some files, such as the Compiler, duplicate links to a single copy of the file is preferable to making individual copies of the file for each workstation. The Filer's Duplicate link command can be used for this purpose, as described in the section, "Duplicating Links to System Files," later in this discussion.

A workstation's SYSTEM directory can also contain the files that "personalize" a workstation, such as customized copies of LIBRARY, INITLIB, AUTOSTART, and so forth.

This directory structure makes booting a smooth and automatic process. With Boot ROM version 3.0 or later (but not 3.0L), a workstation can boot from the SRM with the system to be booted being selected by name at powerup. Thereafter, the workstation looks for the necessary files in its SYSTEM directory (SYSTEM*nn*). If the workstation can't find INITLIB in its SYSTEM*nn* directory, it looks in SYSTEM.

If a workstation boots from the SRM system disc or has no local hard disc on-line, the system volume is identified as unit #45 (prefixed to the workstation's SYSTEM*nn* directory) and the default volume is #5 (another SRM volume, prefixed to the root of the SRM directory structure). Even if the SRM system disc is not chosen as the system volume (using the scheme above), it will still be accessible through units #5 and #45.

The WORKSTATIONS directory should also contain at least one more special directory called TEMP_FILES. All temporary files are created in this directory, and are removed when no longer needed. If you don't create this directory, the first workstation to need it will do so. Consequently the directory WORKSTATIONS should not be write-protected unless directory TEMP_FILES has already been created.

To provide users with a private directory to use as their default volume, you may create a directory called USERS in the root, and within USERS, a private directory for each individual. After booting, users may set the current working directory for their unit #5 to their private directory. A modified TABLE program or an AUTOSTART file can be created to do this automatically. Refer to the chapter on "Setting Up Your Environment" in the *Pascal User's Guide* for information on creating AUTOSTART files. This keeps the root from getting cluttered.

Overview of SRM Installation

Configuring your system to access SRM is not a hard or complicated operation, but it is important that you follow the subsequent procedures in exact detail. Because you are less likely to make mistakes if you understand what's going on, here is an outline of what you will do.

1. Install driver modules DATA_COMM and SRM by executing them (they are programs that install themselves automatically).
2. Execute the TABLE auto-configuration program. When TABLE is executed while the DATA_COMM and SRM driver modules are installed, it finds the SRM system and assigns unit #5 to the SRM system disc.
3. If they are not already on the SRM system disc, create the SYSTEMS and WORKSTATIONS directories at the root of the SRM directory structure, and the SYSTEM directory in the WORKSTATIONS directory.
4. Copy the system boot file (SYSTEM_P) to the SYSTEMS directory. Copy the rest of the Pascal system files to the SYSTEM directory. (The boot ROM expects to find the Pascal system in these directories.)
5. Use the Librarian to create (on the SRM system disc) a new INITLIB file that contains the DATA_COMM and SRM modules, and then replace the existing INITLIB with this new one. (If you have Boot ROM version 3.0 or later, you will be replacing the INITLIB in the SYSTEM directory. With earlier version boot ROMs and Boot ROM 3.0L, you will be replacing the INITLIB on the BOOT: disc.)
6. Re-boot the computer, and verify the new configuration.
7. (Optional) Modify the TABLE program to assign additional unit numbers to the SRM system.

Installing the SRM Driver Modules

First, install the DATA_COMM module. The file is on the CONFIG: disc that is supplied with your system. Although you may have already copied the file onto another volume, such as a local hard disc, this example assumes that you will be loading and executing DATA_COMM from the CONFIG: disc.

Execute the file by pressing [X] at the Main Command Level. The system prompts:

```
Execute what file?
```

Enter this file specification:

```
CONFIG:DATA_COMM. [ENTER] or [Return]
```

Be sure to include the trailing period to suppress the ".CODE" suffix.

Install the SRM module similarly. SRM is also on the CONFIG: disc.

Re-Configuring with TABLE

Execute the TABLE program, which is on the BOOT: disc supplied with your system. Press [X], then answer the `Execute what file?` prompt with:

`BOOT:TABLE.` [ENTER] or [Return]

Again, be sure to include the trailing period.

When the program has finished, you can use the Filer's Volumes command to see that unit #5 is assigned to the SRM system disc. From the Main Command Level, press [F] and then [V]. Here is a typical display:

```
Volumes on-line:
  1   CONSOLE:
  2   SYSTERM:
  3 # BOOT:
  5 # SRM:
  6   PRINTER:
Prefix is - SRM:
```

If the volume name of the SRM system disc (in this example, `SRM`) is not shown in the display, re-execute the DATA_COMM, SRM, and TABLE programs. You may have done something wrong in that process.

If the Filer's Volumes command still does not recognize the #5: volume, check to see whether the SRM hardware is properly configured and installed. For instance, the (unmodified) TABLE program expects the SRM interface in your computer to be set to select code 21.

If that does not work, then you should refer to the troubleshooting sections of the *SRM Operating System Manual.*

Creating Required Directories

To create the directories on your SRM, follow the examples below:

To create the WORKSTATIONS directory, use the following Filer sequence:

1. Press [M] for the Make-directory command. The Filer responds with this prompt:

   ```
   Make file or directory (F/D) ?
   ```

2. Press [D] to make a directory. The Filer responds with the prompt:

   ```
   Make directory (valid only for SRM type units)
   Make what directory?
   ```

3. Respond by typing:

   `#5:/WORKSTATIONS` [ENTER] or [Return]

   to which to the Filer should reply:

   ```
   Directory is 'WORKSTATIONS' correct ? (Y/N)
   ```

4. Press [Y]. The directory is created, and the following message is displayed:

   ```
   Directory WORKSTATIONS made.
   ```

If the computers in the SRM configuration have Boot ROM version 3.0 or later, which allows booting from the SRM, you will also want to create a directory called SYSTEMS at the root. For more information about the SYSTEMS directory, refer to the section on "Planning the SYSTEMS Directory" earlier in this chapter.

Repeat the steps above, answering the `Make what directory?` prompt with:

`#5:/SYSTEMS` (ENTER) or (Return)

You may also wish to create the USERS directory at the root as you did SYSTEMS.

Next, create directory SYSTEM under WORKSTATIONS to store the master copy of all Pascal system programs, such as the Compiler. To reduce the amount of typing involved, you can make WORKSTATIONS the current working directory for unit #5.

5. Press (P) for the Prefix command. The Filer responds:

   ```
   Prefix to what directory ?
   ```

6. Answer by typing:

   `#5:/WORKSTATIONS` (ENTER) or (Return)

   to which the Filer responds:

   ```
   Prefix is WORKSTATIONS:
   ```

To create the SYSTEM, SYSTEM*nn* and TEMP_FILES directories, use the same steps you used to create the WORKSTATIONS directory, answering the `Make what directory?` prompt with the name of the directory to be created. (You need not specify `#5:/WORKSTATIONS`.) Note that creating TEMP_FILES is necessary only if you plan to write-protect the WORKSTATIONS directory.

Copying the System Files to SRM

You may now place the required files into the new directories:

1. To prefix the current working directory to SYSTEM, press (P) for the Prefix command.
2. Type:

   `#5:/WORKSTATIONS/SYSTEM` (ENTER) or (Return)

   To which the Filer responds:

   ```
   Prefix is SYSTEM:
   ```

3. Insert the BOOT: disc in the drive you have been using and copy all the files on it into the new working directory. Press (F) for the Filecopy command. The Filer prompts:

   ```
   Filecopy what file?
   ```

4. Specify that you want all files on the BOOT: disc to be copied by using the = wildcard as follows:

   `BOOT:=,$` (ENTER) or (Return)

   The Filer copies the files one after another.

Repeat the above operation for each of the Pascal system discs (ACCESS:, SYSVOL:, and so forth). After this is done, the SYSTEM directory contains the entire Pascal Workstation system.

Duplicating Links to System Files

To make the system files available in the SYSTEM*nn* directory of each workstation, use the Filer's Duplicate Link command:

1. Press D.

   The Filer responds:

   ```
   Duplicate link (valid only for SRM type units)
   Duplicate or Move ? (D/M)
   ```

2. Press D. The Filer prompts:

   ```
   Dup_link what file?
   ```

3. Answer by typing:

   ?,#5:/WORKSTATIONS/SYSTEM*nn*/$ ENTER or Return

   substituting a two-digit node number for *nn* each time (a leading 0 is required for single-digit node numbers). The "?" wildcard tells the Filer to ask if you want links to each file in the source directory. Answer Y for every file **except** AUTOSTART and SYSTEM_P.

The Duplicate link displays each file name as the links are made.

The last detail is optional. If any of the workstations in the SRM system have Boot ROM version 3.0 or later and you want them to boot from the SRM instead of using local mass storage, you need to put a copy of the Pascal system boot file in the SYSTEMS directory (not in the SYSTEM directory under WORKSTATIONS). The system boot file (SYSTEM_P) is on the BOOT: disc shipped with the system. (You may have already made a backup copy of that disc.) The Duplicate link command links the SYSTEM_P file to the SYSTEMS directory.

1. Press D for the Duplicate link command.

   The Filer responds with:

   ```
   Duplicate link (valid only for SRM type units)
   Duplicate or Move ? (D/M)
   ```

2. Press D.

   The Filer prompts with:

   ```
   Dup_link what file?
   ```

3. Respond by typing:

   #5:/WORKSTATIONS/SYSTEM/SYSTEM_P,#5:/SYSTEMS/$ ENTER or Return

Once the system files are all in the appropriate directories on the SRM system disc, any workstation using the BOOT: disc you create in the next procedure will be able to access the SRM via logical units #5 and #45. If a workstation has high performance local mass storage such as a fixed disc, that workstation's system volume will be on the local mass storage. Otherwise the SRM directory #45:/WORKSTATIONS/SYSTEM*nn* will be the the system volume.

You may wish to also create a working SRM directory within the USERS directory for each user in addition to the SYSTEM*nn* directories for each workstation. A user may then use unit #45 for his system volume and #5 will be prefixed to his working directory.

SRM as the System Volume

You may now designate the workstation's SYSTEM*nn* directory as its system volume. You first need to re-execute the TABLE program to assign unit #45: to this directory. Press [X] at the Main Command Level, and enter this file specification:

/WORKSTATIONS/SYSTEM*nn*/TABLE. [ENTER] or [Return]

replacing *nn* with the node number of the workstation. Don't forget the period.

Now you can execute the Newsysvol command (at the Main Command Level) and specify #45: as the unit number. You may use the What command to verify that all of the subsystems (EDITOR, FILER, etc.) were found in the workstation's SYSTEM*nn* directory. Designating SYSTEM*nn* as the system volume allows you to access the SRM copies of the Pascal subsystems from the workstation by pressing keys such as [E] for Editor, and so forth.

Note

You should not change the prefix on unit #45 once it is assigned to SYSTEM*nn*.

Adding Modules to INITLIB

This procedure adds the DATA_COMM and SRM modules (shipped on the CONFIG: disc) to INITLIB (shipped on the BOOT: disc), creating a new INITLIB on the SRM system disc that includes the drivers required for the SRM.

1. At the Main Command Level, press [L] to load the Librarian (note that the Librarian should be loaded from the SRM system disc).
2. When you see the Librarian's prompt line at the top of the CRT, press [O] to specify the name of the (Output) file the Librarian will be creating (called INITNEW).
3. Type:

   #5:/WORKSTATIONS/SYSTEM/INITNEW [ENTER] or [Return]
4. Press [I] so you can specify an Input file, then type:

   #5:/WORKSTATIONS/SYSTEM/INITLIB. [ENTER] or [Return]

   Be sure to type the period after the word INITLIB in this command (to suppress the .CODE suffix). The Librarian responds by showing INITLIB as the name of the input file.
5. Near the bottom of the CRT a line appears that says:

   M input Module: KERNEL

   Press [T] to transfer this module to the Output file. After a few moments, the name of a new module (KBD) appears. Each time a new module name appears, press [T] to move it to the output file. You should continue copying modules until the name LAST appears. **Do not copy the module LAST yet.**
6. Now you must get the required SRM and DATA_COMM drivers and include them in INITNEW. Press [I] and type:

   #5:/WORKSTATIONS/SYSTEM/DATA_COMM. [ENTER] or [Return]

   Don't forget the period after the name.

7. When the module name DATA_COMM appears near the bottom of the screen, press [A] to tell the Librarian to transfer the module to the Output file (INITNEW).
8. Press [I] and type:

   #5:/WORKSTATIONS/SYSTEM/SRM. [ENTER] or [Return]

   again, being sure to include the period.
9. When the module name SRM appears on the bottom of the screen, press [A] to transfer SRM to INITNEW.
10. Press [I] and type:

    #5:/WORKSTATIONS/SYSTEM/INITLIB. [ENTER] or [Return]
11. When module name KERNEL shows up near the bottom of the screen, select module LAST instead by pressing [M], then typing:

    LAST [ENTER] or [Return]
12. Transfer LAST to INITNEW by typing [T].
13. You now have all the modules in your new library. "Keep" INITNEW by typing [K], then quit the Librarian by typing [Q].

Replacing INITLIB

Where you place the new version of INITLIB depends on which boot ROM is in your machine.

- If you have Boot ROM version 3.0 or later (but not 3.0L), you will probably want to leave the modified INITLIB in the SYSTEM directory, where it will be found automatically when workstations boot from the SRM.
- If you have an earlier version boot ROM or Boot ROM 3.0L, you need to replace the INITLIB on the BOOT: disc with the new INITLIB (INITNEW) because these boot ROMs cannot boot directly from the SRM – they must use the BOOT: disc.

With Boot ROMs Version 3.0 and Later

1. Use the Filer's Change command to re-name the existing INITLIB (in /WORKSTATIONS/SYSTEM) to something like OLDINITLIB.
2. Use the Change command again to re-name the INITNEW file to INITLIB.
3. Re-boot your workstation to verify that the new INITLIB file works correctly.
4. Use the Filer's Duplicate link command to link the new INITLIB to the SYSTEM*nn* directories of the workstations that will be booting from the SRM. (Or you may make custom INITLIB files for each workstation.)

With Earlier Version Boot ROMs

1. Press [F] to invoke the Filer.
2. Put the spare copy of the BOOT: disc (**not** the original) into a drive. Press [R] for the Remove command. The computer responds with:

   Remove what file?
3. Answer by typing:

   BOOT:INITLIB [ENTER] or [Return]

   Note that there is no period after the file name this time.

4. Press [K] (Krunch) to pack all the remaining files on the disc to make the maximum amount of room for the new INITLIB. The Filer prompts:

```
Crunch what directory?
```

5. Answer by typing:

   BOOT: [ENTER] or [Return]

   The Filer prompts:

```
Crunch directory BOOT ? (Y/N)
```

6. Answer by pressing [Y]. The Filer then prompts:

```
Crunch of directory BOOT in progress
DO NOT DISTURB!!
```

Note

If you interfere with the disc before the crunch operation completes, you will ruin the data on the disc. You will certainly have to recopy it from the original BOOT: and you may have to re-initialize it.

After the crunch is complete, the Filer prompts:

```
Crunch completed
```

7. You may now Filecopy the INITNEW file onto the spare BOOT: disc and change the file's name to INITLIB. To do so, insert the spare BOOT: disc into a disc drive and press [F] for the Filecopy command.

```
Filecopy what file?
```

   Answer:

   #5:/WORKSTATIONS/SYSTEM/INITNEW.CODE,BOOT:INITLIB [ENTER] or [Return]

   When the Filecopy finishes, you have a BOOT: disc on which the INITLIB contains the SRM drivers.

8. Verify that the new INITLIB works by re-booting your system from that disc.

Each Pascal workstation in the system with earlier (or 3.0L) boot ROMs must boot using an INITLIB that includes the SRM driver software. You may wish to make, for each of these workstations, a copy of the disc you've just created for each workstation.

Multi-Disc SRM

When an SRM system has more than one hard disc, you need to modify, recompile, and execute the CTABLE program to allow access to these discs. This section describes how to perform this type of configuration change.

When more than one shared disc is installed on the SRM system, each disc must have a WORKSTATIONS directory. If the directory is write-protected, a TEMP_FILES directory must be created. You may also wish to create another SYSTEMS directory. Boot ROMs version 3.0 and later search for bootable systems on each disc containing a SYSTEMS directory.

CTABLE Modifications

Near the end of the CTABLE program, just above the manual `templates` section, a small section of code assigns Unit Table entries for the SRM.

```
with SRM_dav do
  begin
    {  tea_srm( 46, sc, ba, du); {free}
    tea_srm( 45, sc, ba, du);  {for possible use as the system unit}
  end; {with}
```

The first `tea_srm` entry provides a template for assigning unit #46: to the second disc connected to the SRM. To assign that unit, replace the `du` parameter with the second disc's volume address.

To find the correct volume address, use the VOLUMES command at the SRM system controller. The resulting display lists the volume address of each volume configured to the system. (With the SRM 1.0 operating system, the volume address is shown under the column labeled `unit #`. With SRM 2.0, the column is labeled `vol add`.)

To activate the first `tea_srm` statement, you must remove the comment delimiter ({) from the beginning of that line.

Just below the manual "templates" section of the CTABLE program is another section pertaining to units for the SRM.

```
{ prefix the primary and secondary SRM unit entries }

if not unit_prefix_successful('#5:/') then {do nothing};
{tries to set up uvid for possible default unit assignment below}

{ if not unit_prefix_successful('#46:/?') then zap_assigned_unit(46);  {free}

if not unit_prefix_successful('#45:'+srmsysprefix+srmnode(unitable^[45],sc)) then

if not unit_prefix_successful('#45:'+srmsysprefix) then
  zap_assigned_unit(45);
```

If you remove the leading comment delimiter ({) from the `#46:` entry and remove the question mark from the literal `'#46/?'`, Pascal will be able to recognize the second hard disc connected to the SRM system.

If you wish to have a Unit Table entry for a particular directory, you can identify the directory by including its path in the specification. For example:

```
if not unit_prefix_successful('#46:/USER/AL') then zap_assigned_unit(46);
```

If you make this modification be sure to activate both this statement and its accompanying `tea_srm` procedure `tea_srm( 46, sc, ba, du); {free}` by removing the comment delimiter ({) from the beginning of each of the program lines.

In this example, this modification causes the system to boot with unit #46 assigned to the directory "/USER/AL" on the SRM disc selected by the first `tea_srm` statement.

After all modifications have been made, you can compile CTABLE. Remember that you need to enable the `$search 'CONFIG:INTERFACE'$` Compiler option at the beginning of the program and make the INTERFACE library accessible at compile time. You will probably also want to link the resultant TABLE object file to itself with the Librarian to conserve disc space. See the procedures in the "Modifying the TABLE Program" section of the *Pascal Workstation System* manual for explicit details.

Glossary

Appendix A

directory name A directory name is the same as a remote file name because a directory is a type of remote file. Directory names consist of from one to 16 characters, including uppercase and lowercase letters, the digits 0 through 9, the underbar (_) character, the period (.) character, and ASCII characters decimal 161 through 254.

file name The term "file name" refers to local files, not to files on the SRM system (described under "remote file name"). A file name consists of one to 10 characters. HP Series 200 file names can contain uppercase and lowercase letters, the digits 0 through 9, the underbar (_) character, and ASCII characters decimal 161 through 254.

name With the BASIC language system, a name (used in I/O path name) consists of one to 15 characters. The first character must be an uppercase ASCII letter or one of the ASCII characters decimal 161 through 254. The remaining characters, if any, can be lowercase ASCII letters, digits, the underbar (_), or ASCII characters decimal 161 through 254.

Names may be any combination of uppercase and lowercase letters, but may not look like a keyword. Conflicts with keywords are resolved by mixing the letter case in the name.

node address An integer from 0 through 63 that identifies an SRM device (such as a workstation or controller).

password Passwords are used to protect access to remote files and directories. Passwords consist of one to 16 characters. All ASCII characters except " > " are allowed. Passwords are assigned by the PROTECT statement in BASIC or the Pascal Filer's Access command.

remote file name A remote file name consists of one to 16 characters. HP Series 200 remote file names can contain uppercase and lowercase letters, the digits 0 through 9, the underbar (_) character, the period (.) character, and ASCII characters decimal 161 through 254.

SRM The acronym for Shared Resource Management.

SRM controller The HP Series 200 computer that controls access to the shared resources of the Shared Resource Management system.

SRM controller's node address An integer in the range 0 through 63 that identifies the SRM controller.

SRM interface The term used to describe the HP 98629A Resource Management Interface resident in an SRM workstation computer (not the interface in the SRM controller).

volume A named portion of mass storage media, which may contain several files. Disc drives supported by HP Series 200 mass storage operations contain only one volume per disc.

volume name A name used to identify a mass storage volume. The volume name is assigned to the volume at initialization. Volume names consist of one to 16 characters including uppercase and lowercase letters, the digits 0 through 9, the underbar (_) character, the period (.) character, and ASCII characters decimal 161 through 254.

volume password A "master" password, assigned at initialization, that allows complete access to all files on a mass storage volume. Volume passwords consist of one to 16 characters. All ASCII characters except " > " are allowed. The volume password supercedes all access restrictions placed on files by the PROTECT statement in BASIC or the Pascal Filer's Access command.

SRM Interface STATUS Registers

Appendix B

Status Register 0 Card Identification

52 if the Remote Control switch (R) is set to 0 (closed); 180 if switch is set to 1 (open).

Status Register 1 Interface Interrupts

1 = interrupts enabled; 0 = interrupts disabled.

Status Register 2 Interface Busy

1 = busy; 0 = not busy.

Status Register 3 Interface Firmware ID

Always 3 (the firmware ID of the HP 98629A interface).

Status Register 4 Not Implemented

Status Register 5 Data Availability

0 = receiver buffer empty;
1 = receiver data available but no control blocks buffered:
2 = receiver control blocks available but no data buffered;
3 = both control blocks and data available.

Status Register 6 Node Address of the interface

Node address of the HP 98629A interface installed in **this** computer which is set to the specified select code. The range of node addresses is 0 through 63.

Status Register 7 CRC Errors

Total number of cyclic redundancy check (CRC) errors detected by the interface since powerup or (RESET).

Status Register 8 Buffer Overflows

Total number of times the receive buffer has overflowed since powerup or (RESET).

Status Register 11 Amount of available space (number of bytes) in the transmit-data buffer.

Status Register 12 Number of transmission retries performed since powerup or (RESET).

SRM and BASIC 2.0

Appendix C

This manual describes SRM support of the BASIC 3.0 language system. This appendix summarizes the differences between use of SRM with the BASIC 2.0 language system and SRM use with BASIC 3.0.

The primary differences lie in the startup procedures required for bringing up an HP Series 200 BASIC workstation on the SRM for the first time, and in additional language features implemented in BASIC 3.0 (and therefore not supported with BASIC 2.0).

The first section of this appendix, **System Startup From a BASIC 2.0 Workstation**, describes the initial startup procedure for use with a BASIC 2.0 workstation, and is needed only by a system manager installing an SRM system.

The second section, **BASIC 2.0 Language Features**, summarizes, by keyword, aspects of BASIC 2.0 keyword use that differ from the descriptions in this manual's "BASIC Language Reference" section.

The final section, **Additional BASIC 2.0 Information**, notes all other portions of the main text of the manual that contain information inappropriate or invalid for BASIC 2.0 use on SRM. This section also contains information on modifications to existing programs required by BASIC 2.0 that are additional to those required for use with BASIC 3.0.

System Startup From a BASIC 2.0 SRM Workstation

The following instructions aid the SRM sytem manager in bringing up the SRM system on an HP Series 200 BASIC 2.0 workstation for the first time. These instructions assume that the workstation has its own floppy disc drive.

For more information about the directories and files discussed in this procedure, refer to the "System Startup" chapter in this manual.

To bring up the **first** BASIC 2.0 SRM workstation on your SRM system for the **first** time, follow these steps:

1. Load into the workstation the BASIC operating system software required for accessing SRM.

   Before proceeding, you should have installed the SRM operating system on the system controller by completing steps one through eight in the Initial Installation section of the "System Installation" chapter of the *SRM Operating System Manual.*

   a. With the computer's power OFF, insert the BASIC 2.0 system disc into the workstation's primary drive (see the operating manual for the workstation or disc drive to determine which is the primary drive) then turn the computer's power ON. Allow the BASIC operating system to completely load, then remove the disc.

   b. Insert the Resource Management Access disc (containing the SRM2_1 BIN file) into the primary disc drive and type:

      ```
      LOAD BIN "SRM2_1"
      ```
      EXECUTE

      If the BIN file is on a disc in an external (non-SRM) drive, you must either ensure that the device is the current MASS STORAGE IS device or include the appropriate mass storage unit specifier (msus) in the file specifier. Refer to the "Data Storage and Retrieval" chapter of the *BASIC Programming Techniques* manual for further information on msus.

2. Establish the remote device (shared disc) as the workstation's mass storage device by typing:

   ```
   MSI ":REMOTE"
   ```
   EXECUTE

   which establishes the root of the SRM directory structure (on the shared disc) as the workstation's current working directory. You may verify that position within the directory structure by using the CAT command.

3. Create the SYSTEMS and USERS directories on the shared disc by typing:

   ```
   CREATE DIR "SYSTEMS"
   ```
   EXECUTE

   to create the SYSTEMS directory, and:

   ```
   CREATE DIR "USERS"
   ```
   EXECUTE

   to create the USERS directory.

4. Install the SRM operating system (from the controller) in the newly-created SYSTEMS directory:

   a. **Return to the "System Installation" chapter** of the *SRM Operating System Manual* and resume the Initial Installation procedure from step 9.

   b. After completing that procedure, return to the workstation you used to create the SYSTEMS directory and continue with the following steps.

5. Place BASIC operating system files into the SYSTEMS directory, following these steps:

   a. Specify SYSTEMS as the current working directory by typing:

      ```
      MSI "SYSTEMS"
      ```
      EXECUTE

   b. Place the disc containing the BASIC Loader Utility into the workstation's primary drive and type:

      ```
      COPY "SYSTEM_LD:<device specifier> TO "SYSTEM_LD"
      ```
      EXECUTE

      where *<device specifier>* might be `HP8290X,700,0` on the HP Models 220 and 216 and is usually `INTERNAL` on the HP Models 226 and 236.

      To allow workstations to boot automatically from the SRM, SYSTEM_LD should be the first system file stored in SYSTEMS after the SYSTEM_SRM and CONFIG_SRM files.

   c. Remove the Loader Utility disc from the workstation's primary drive and insert the disc containing the BASIC operating system. Type:

      ```
      COPY "SYSTEM_BAS:<device specifier> TO "SYSTEM_BAS"
      ```
      EXECUTE

      After the copy is complete, remove the disc from the drive.

   d. (Optional) You may wish to copy the AP2_1 BIN file and other BIN files from the Extended BASIC 2.1 disc. To copy the AP2_1 BIN file, with the Extended BASIC 2.1 disc in the workstation's primary drive, type:

      ```
      COPY "AP2_1:<device specifier> TO "AP2_1"
      ```
      EXECUTE

      Use the COPY statement as illustrated above to copy other BIN files from the Extended BASIC 2.1 disc then remove the disc from the drive.

   e. With the Resource Management Access disc in the workstation's primary drive, type:

      ```
      COPY "SRM2_1:<device specifier> TO "SRM2_1"
      ```
      EXECUTE

6. Create configuration files for use by the BASIC Loader Utility to allow workstations to boot automatically from the SRM. Use of the Loader Utility allows workstations with Boot ROM version 3.0 or later (but not Boot ROM 3.0L) to boot automatically from the SRM. For more information, refer to the *Loader Utility Manual.*

   a. With SYSTEMS as the current working directory (see step 5a), type:

      ```
      LOAD "CONFIGER:<device specifier>
      ```
      EXECUTE

   b. Press RUN to create the default configuration file. The screen displays the prompt:

      ```
      Enter CONFIG file name
      ```

      to which you reply:

      ```
      CONFIG_LD
      ```
      ENTER

Note

For workstations whose boot ROMs support automatic booting from SRM, at powerup the boot ROM boots the Loader Utility (assuming SYSTEM_LD is the first bootable system file in the SYSTEMS directory).

The Loader Utility then loads the system and BIN files specified either by the workstation's CONFIG_LD*nn* file (where *nn* is the workstation's node address) or by the default configuration file CONFIG_LD if the appropriate CONFIG_LD*nn* does not exist.

c. To specify the BASIC 2.0 operating system as the system to be booted into the workstation, type:

```
SYSTEM_BAS  CONTINUE
```

You may also wish to add a key string to the file. The key string is described in the *Loader Utility Manual.* You may continue to enter names of non-scratchable BIN files you wish to load, the last of which should be SRM_AP2_1.

Note

If you decide to load SRM_AP2_1 via a configuration file, you must make SRM_AP2_1 the last entry in the file. You must also follow this entry with a blank line.

When the workstation is powered up and the SRM_AP2_1 BIN file loads, a non-fatal error occurs. This is normal and should not affect the operation of the SRM system or the workstation or the execution of any key string included in the configuration file.

d. Repeat steps 6b and 6c above to create unique configuration files for each workstation. When naming the file (see step 6b), include the workstation's node address in the file name. For example:

```
CONFIG_LD10
```

would be the configuration file for the workstation at node address 10.

BASIC 2.0 Language Features

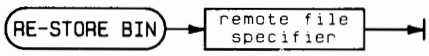
The following table lists keywords used differently on SRM with BASIC 2.0 than with BASIC 3.0 and describes any differences from the keyword's description in this manual's "BASIC Language Reference" section. If a keyword is not listed in this table, you may use that keyword with either version of BASIC, exactly as described in this manual.

One major difference is in the BIN files supporting the use of several of the keywords. With BASIC 2.0, the SRM_AP2_1 (SRM Advanced Programming) BIN file provides enhancements and additions to the BASIC 2.0 command statements supported on SRM. SRM_AP2_1 requires the BASIC 2.0 AP2_1 and the SRM2_1 BIN files.

Keyword	Differences
CAT	Options (SELECT, SKIP, RETURN, NO HEADER, PROTECT, TO String_array$(*), <PROG file>) require the SRM_AP2_1 BIN file.
	The catalog listing header shows only the name of the directory or file being listed, not the full path to that directory or file, as in illustrations (valid only for BASIC 3.0).
LOAD BIN	BIN files **cannot** be loaded from an external device unless the BIN file to access that device is already present in the workstation.
LOAD KEY	Requires SRM_AP2_1 BIN file.
LOADSUB	All versions of this command **except** LOADSUB ALL FROM ... require the SRM_AP2_1 BIN file.
PLOTTER IS	Not supported with BASIC 2.0. Cannot be used with the SRM 1.0 operating system.
PRINTER IS	Not supported with BASIC 2.0. Cannot be used with the SRM 1.0 operating system.
RESET	Requires SRM_AP2_1 BIN file.
RE-STORE BIN	A valid statement in BASIC 2.0 (not in BASIC 3.0). See description on the following page.
RE-STORE KEY	Requires SRM_AP2_1 BIN file.
STORE	This statement creates a PROG file and stores the currently resident BASIC program and *all normal BIN files currently in memory.*
STORE BIN	A valid statement in BASIC 2.0 (not in BASIC 3.0). See following description.
STORE KEY	Requires SRM_AP2_1 BIN file.
STORE SYSTEM	Not a valid BASIC 2.0 statement.
SYSTEM$	Only SYSTEM$("MSI") is valid with BASIC 2.0. Does not return the full directory specifier (including directory path) as with BASIC 3.0. Returns only the directory's name and full remote msus.
TRANSFER	Requires SRM_AP2_1 BIN file.

RE-STORE BIN

With SRM, RE-STORE BIN creates a BIN file in a remote directory and stores all normal BIN files in the file. For a description of RE-STORE BIN with local files, refer to the *BASIC 2.0 Language Reference.*



Example Statements

```
RE-STORE BIN "Phyrec"
RE-STORE BIN "Dir<RWpass>/Bin_prog<RWpass>"
```

Semantics

READ and WRITE access capabilities are required on the directory immediately superior to that in which the file is to be re-stored and on the BIN file (if it exists).

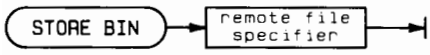
Attempting to use RE-STORE BIN with an existing file that is not a BIN file results in an error. Passwords assigned to an existing file are retained when a RE-STORE BIN is performed on the file. If you specify the wrong password on a protected file, the system returns an error message.

If the file does not already exist, RE-STORE BIN performs the same action as STORE BIN. Including a password in the RE-STORE BIN statement's remote file specifier does not protect the file. Passwords are assigned only with PROTECT.

With SRM, temporary files may by left on the disc by RE-STORE BIN if (CLR I/O) or (RESET) are pressed or a TIMEOUT occurs during the RE-STORE BIN operation. The name of the temporary file is an eight-digit number derived from the value of the workstation's real-time clock when the interruption occurred. You may wish to check the contents of any such file before purging.

STORE BIN

With SRM, STORE BIN creates a BIN file in a remote directory and stores all normal BIN files currently resident in memory in the file. For a description of STORE BIN with local files, refer to the *BASIC 2.0 Language Reference.*



Example Statements

```
STORE BIN "FFT"
STORE BIN "Dir<RWpass>/Bin_prog"
```

Semantics

READ and WRITE access capabilities are required on the directory immediately superior to that in which the file is to be stored.

STORE BIN opens the remote file in exclusive mode (denoted as LOCK in a CAT listing) and enforces that status on the file until the STORE BIN is complete. While in exclusive mode, the file is inaccessible to all SRM workstations other than the one executing the STORE BIN.

Including a password in the STORE BIN statement's remote file specifier does not protect the file. Passwords are assigned only with PROTECT.

Additional BASIC 2.0 Information

The following information in the main text of this manual is invalid for BASIC 2.0 use of SRM (all section titles refer to the chapter on "HP Series 200 BASIC Workstation Use on SRM"):

To execute statements in BASIC 2.0, use the (EXECUTE) key on the HP 98203B keyboard or the key labeled (EXECUTE) on the HP 98203A keyboard. The (ENTER) key is **not** an execution key for BASIC 2.0 commands and statements. BASIC 2.0 does not support the HP 46020A keyboard.

Throughout the section, "Using Your BASIC Workstation on SRM,"

- All example catalog listings show the header format used with BASIC 3.0. With BASIC 2.0, the header lists only the name of the directory rather than the full path name to the directory.

  Similarly, SYSTEM$("MSI") returns the full directory specifier (including the path to the directory) with BASIC 3.0, but only the current directory's name with BASIC 2.0.

In the section, "Booting From the SRM,"

- BASIC 2.0 does not allow you to load BIN files from a device into the workstation unless the BIN file to access that device is present.
- The Loader Utility can be used to load BIN files from SRM with BASIC 2.0 but not with BASIC 3.0. You must use the Loader Utility to boot BASIC 2.0 into your workstation from SRM.
- The SYSBOOT command is not a valid BASIC 2.0 command.
- BASIC 2.0 does not recognize "autostart" files if you are using the Loader Utility. All automatic configuration is specified in the Loader Utility's configuration files (CONFIG_LD or CONFIG_LD*nn*).
- In the BASIC 2.0 remote msus, the default value for the interface select code of a workstation's SRM interface (HP 98629A) is 21. The default controller's node address is 0.

Modifying Existing Programs to Access Shared Printers or Plotters

Spooling

With BASIC 2.0, programming for printer and plotter spooling requires replacing the PRINTER IS or PLOTTER IS statements with routines that implement those functions.

For example, in BASIC 2.0, you cannot use the sequence:

```
PRINTER IS ":REMOTE"
PRINT The_data$
```

to spool data to the shared printer. Instead, you must create a file in the spooler directory and output strings to the file. After the file is closed, the SRM controller automatically places the file in a "waiting list." The SRM controller locks the file while it is being printed, then purges the file when printing is completed.

Sending information to a spooler file may require four main changes to a program:

- Replacing all PRINTER IS statements with a CALL to a printer-selecting subprogram that assigns an I/O path name to the designated device or file.
- Replacing all PRINT statements with OUTPUT statements to access the printer to which the I/O path name has been assigned. For example:

```
500 OUTPUT @Printer;"Final Results"
```

- If double-spaced output is not desired, changing all numeric values to string values with the VAL$ function. For example:

```
510  FOR Place=1 TO 10
520    OUTPUT @Printer;VAL$(Place)&"  "$Name$(Place)
530  NEXT Place
```

  (Otherwise a carriage return/line feed is sent after each item in the OUTPUT statement.)

- When a printing segment of the program is done, closing the I/O path (using, for example, ASSIGN @ ... TO *) to allow the SRM controller to print the file's contents. The spooler will not print the contents of an open file.

Formatted Output

Formatted output (OUTPUT...USING...) cannot be directed to spooler files, which are ASCII files. However, formatting can be accomplished by declaring a string variable, outputting to the string, and then outputting the string to the spooler file. The following program shows an example of this technique:

```
100   DIM Output$[80]
      .
      .
      .
1500  ! Center title by using image specifiers.
1510  OUTPUT Output$ USING "#,34X,K";"Final Results"
1520  OUTPUT @Printer;Output$
1530  !
1540  FOR Place=1 TO 10
1550    OUTPUT Output$ USING 1560;Place,Name$(Place)
1560    IMAGE #,DD,5X,40A
1570    OUTPUT @Printer;Output$
1580  NEXT Place
```

Note

If the EOL-suppression image specifier (#) is not used, a carriage return and line feed will be printed as the last two characters in the string variable, causing double-spaced output.

Subject Index

a

access capabilities 3, 16, 25, 57
access capability requirements 32
access, concurrent 78
Access command (Pascal Filer) 33, 77, 87, 89, 91
access rights 77, 89
ALL attribute option 89
ASSIGN 5, 15, 41, 46, 51, 54, 62, 66, 70, 71
  and locked files 35
attributes, password protection 77
autostart file (BASIC) 8, 98, 103, 128

b

BASIC 2.0 (system startup) 122
BASIC language, SRM support of 1
BDAT files (sending to spooler) 16
bootable system file 98
booting from SRM 6, 97, 106, 110, 123
boot ROM versions 104
Boot ROMs 3.0 or later 6, 65, 76, 97, 106, 109, 110, 112, 123

c

CAT 9, 11, 36, 125
catalog listing (BASIC) 10, 128
  format of 37
  header 37-38, 128
CHECKREAD 40
closing remote files 20, 55, 56, 59
CLR I/O 61, 63, 126, 126
CONFIG_SRM 96
configuration file, Loader Utility 100, 102
CONTROL 41
controller's node address 8, 29, 74
  default value 9
COPY 42
copying files 12
  using COPY 12, 101
  using ENTER and OUTPUT 13
  using LOAD and STORE 13
corrupt 78
CREATE ASCII 43
CREATE BDAT 44
CREATE DIR 1, 45
creating files and directories 10
CTABLE program, Pascal 74, 113
  modifications to 114-115

d

DATA_COMM driver module (Pascal) ... 107, 111
DCOMM BIN file 7, 95, 97
default volume (Pascal) .. 75, 82, 86, 106, 107
DELETE 58
directories 2
  capabilities of 3
  closing 19, 20
  creating 9, 85
  opening 45
  protecting 3, 16
  removing 19
  subordinate 2
  superior 2
directory name 27
directory path 3, 27, 76
  syntax of (Pascal) 87
directory specification (Pascal) 86
directory specifier (BASIC) 25, 30
Duplicate link command, Pascal Filer 79, 91, 110
duplicate links 106

e

ENTER 46
error codes 72
exclusive mode 47, 61, 64, 67, 127
extent 43, 44, 45, 88

f

file size specification (Pascal) 88
file specification (Pascal) 86

g

GET 47

h

hierarchical directory structure 2, 75, 76
HP 98629A interface card 74

i

INITIALIZE 48
INITLIB 106, 107, 111, 112
I/O path, closing 35, 51, 59, 71, 129
I/O path name (see ASSIGN) 35

k

key references 6, 80, 94, 104, 128

l

LABEL 29, 30
Librarian (Pascal) 111
List directory command, Pascal Filer 82
LOAD 49
Loader Utility (BASIC) 99, 123, 124, 128
LOAD BIN 49, 125
LOAD KEY 49, 125
LOADSUB 50, 125
LOCK 1, 5, 20, 47, 51, 71
Locking files (BASIC) 5

m

Make command, Pascal Filer 85, 91,108
MANAGER access capability (BASIC) 5, 38, 57
MASS STORAGE IS (MSI) 19, 45, 52
msus, local 12, 101
msus, remote vs. local 22

n

non-contiguous storage of files 4
node address (SRM workstation) 74, 104

o

ON TIMEOUT 53
operating system, selecting 7
OUTPUT 54

p

Pascal Filer commands 91
password 23, 26, 57, 77, 87, 89
password protection 4, 57, 77, 87, 89
PLOTTER IS 55, 125, 128
plotters, preparing 16
Prefix command, Pascal Filer . . 80, 81, 91, 109
PRINTER IS 56, 125, 128, 129
PROTECT, CAT option 1, 39
PROTECT statement 1, 4, 57
protecting files and directories 3, 16
public access 57, 77
PURGE 19, 59
purging files and directories 19

r

READ access capability (BASIC) 4, 38, 57
remote device 8
remote file name 26
remote file specifier 25, 26, 34
remote msus 8, 28, 52
remote msus, generic 29
RENAME 61
RE-SAVE 47, 53
RESET 62, 125
(RESET) 20, 51, 55, 59, 61, 63, 71, 127
RE-STORE 49, 53, 63
RE-STORE BIN 125, 126
RE-STORE KEY 63, 125
root 2, 31, 76, 86, 95, 103, 107

s

SAVE 47, 61, 64
SCRATCH A 20, 51, 55, 56, 65, 71
secondary loader 99
shared mode 47, 49, 50
shared printer, sending program output to .. 15
shared printers and plotters 14, 90
  SRM management of 5
space allocation for remote files 4
spooler directory 5, 14, 15, 55, 56, 75, 90
  writing files to 15
spooling 5, 128
  using PRINTER IS and PLOTTER IS 14
SRM BIN file 7, 49, 95, 97
SRM driver module (Pascal) 107, 111
SRM interface select code 8, 29, 74
  default value 9
SRM operating system, version 1.0 .... vii, 16, 90, 105, 114
SRM operating system, version 2.0 vii
SRM system controller 74
SRM_AP2_1 BIN file 125
SRM2_1 BIN file 122, 125
STATUS 66
STORE 49, 63, 67, 125
STORE BIN 126, 127
STORE KEY 67, 125
STORE SYSTEM 68, 97, 125
subordinate directory 2
superior directory 2
SYSBOOT 7, 128
SYSTEM directories (Pascal) ... 105, 109, 112
system volume (Pascal) 75, 84, 106, 110, 111
SYSTEM_LD 99, 100
SYSTEMS directory 76, 94, 95, 96, 101, 107, 122
SYSTEM_SRM 96
SYSTEM$ 69, 125, 128

t

TABLE program (Pascal) 108, 111
TEMP_FILES 105, 106, 109, 113
TRANSFER 70, 125

u

Unit directory command, Pascal Filer 80, 83, 91
unit numbers (shared discs) 75
UNLOCK 1, 5, 20, 51, 71
USERS directory 94, 95, 96, 109, 122

v

volume address 114
volume identifier, syntax of (Pascal) 86
volume name 29
volume password 29, 52, 87
Volumes command, Pascal Filer 81, 108

v

working directory (BASIC) 3, 10, 11, 31, 59, 83
working directory (Pascal) 86
WORKSTATIONS directory 95, 105, 106, 107, 108
WRITE access capability (BASIC) ... 4, 38, 58

Part No. 98619-90051
E 0584
Microfiche No. 98619-99051

Printed in U.S.A.
First Edition...May 1984