BRIDOS OPERATING SYSTEM
VERSION 1.52
USER'S GUIDE AND INTERFACE MANUALS
December 5, 1983

PROSOFT AB
VERKSTADSGATAN 21
S-117 36 STOCKHOLM
SWEDEN
Tel. Int+46-8 68 09 55

BRIDOS OPERATING SYSTEM VERSION 1.52
USER'S GUIDE

TABLE OF CONTENTS

1.0 GENERAL DESCRIPTION

BRIDOS is a high-speed operating system for microcomputers. It was developed by BRITECH AB of Stockholm, Sweden.

BRIDOS includes the BRISAM file-management system, which is based on tree-structured files (B+ tree). It has been optimized for data-processing applications in single- as well as multi-user systems and offers big-computer capabilities in microcomputer format. The major benefits of the system include:

* Faster data-processing.
* Safer data-processing.
* Logical and dynamic allocation of disk space
* More efficient use of disk space.
* Protection against obsolescence.
* Facilities for volumes or files of up to 64 Mbytes.

BRIDOS and BRISAM offer unique data safety. The risk of lost data and crashed files is virtually eliminated.

BRIDOS and BRISAM generate substantial reductions in run time for all types of programs, from word processing, bookkeeping and inventory management to production control and engineering.

The built-in capability for high-speed search in files as large as 64 Mbytes makes BRIDOS ideal for applications such as administration of hospital records or operation of comprehensive direct-mail systems.

Emulator technology

BRIDOS is designed to accept modular emulators of other operating systems. An emulator creates an environment that is for practical purposes identical to the normal environment of a specific operating system. The BRIDOS CP/M^{R} emulator thus loads into memory and creates a CP/M environment. A call to CP/M is instantaneously translated into a call to BRIDOS. Commands in the application program are executed by the BRISAM file-management system instead of the CP/M file manager. A program written for CP/M runs under BRIDOS at higher speed and with minimal risk of data loss, while the user retains access to the wide variety of software written for CP/M. Emulators are being added for other operating systems, including MSDOS^{R} and CP/M-86^{R}.

1.01 HARDWARE REQUIREMENTS FOR THREE TYPES OF BRIDOS SYSTEMS

Uni-System

This system is designed for single-user applications. It requires a 64 KB byte working memory, a Z80 processor, two disk drives and a video display. The memory area available for CP/M applications is 43 KB, which corresponds to a 48 KB CP/M system. The Uni-System does not support a hard disk or multiple volumes on a single disk.

Omni-System

This system requires the same hardware as the Uni-System and an extra memory bank of at least 32 KB. The memory area available for application programs is 58 KB, which corresponds to a 62 KB CP/M system.

The Omni-System can be used in a single microcomputer or in a network environment, where it is connected to a central data server with a BRIDOS Multi-User System. This provides both local and centralized disk access, i.e. distributed data processing.

Multi-System

The Multi-user BRIDOS System supports multiple hard disks and is designed for applications that do not require local disk access. It is also used in networks together with the Omni-System.

Current applications

All three types of BRIDOS systems are now running on TELEVIDEO microcomputers in Scandinavia, and on IBEX microcomputers (the Japanese micro market leader in Europe) in England, France, West Germany, Switzerland and Scandinavia.

In Sweden, a set of application programs sold under various names is now running under BRIDOS. This system consists of fully integrated modules for incoming orders, invoicing, purchasing, inventory management, accounts payable, accounts receivable, cost accounting, vendor accounts, bookkeeping and auditing. The DAMAPS system for material and production control is also running under BRIDOS; installations include a subcontractor to SAAB and VOLVO.

BRIDOS is also being implemented on 6 other machines, two of which have a combined market share of over 50% in Sweden.

1.02 COBOL REPLACEMENT LIBRARY PACKAGE

A replacement library package is now available for programs written in MICROSOFT COBOL-80. This package enables standard COBOL-80 programs to run on 8-bit microcomputers at extremely high speeds. No modification is required.

In Switzerland, NCR Mini-Micro (a subsidiary of National Cash Register) runs standardized COBOL-80 mainframe programs under BRIDOS. No major program changes were required for implementation under BRIDOS, and performance is close to mainframe levels.

A replacement library package is being developed for MICROFOCUS LEVEL-2 COBOL.

1.03 INTERFACE MODULES

Interface modules are available for MICROSOFT's BASIC Compilers and PL/1 from Digital Research.

1.04 SYSTEM OVERVIEW

The operating system is the fundamental program for a microcomputer. Without an operating system, the microcomputer cannot do its job. Microcomputer performance depends on operating system performance. The faster the operating system, the faster the computer works.

BRIDOS uses a file-management system called BRISAM, which is designed for faster data processing on microcomputers.

BRISAM is also designed for safer data processing. Files are updated while a program is running. Updating is so fast that a power failure or an operator error does not result in loss of data.

Disk space is allocated dynamically in logical volumes, which ensures more efficient utilization of disk space. The area occupied by a file grows or shrinks as data is inserted or removed. BRIDOS volumes are organized in a hierarchical structure which resembles the one in the UNIX system.

BRIDOS enables microcomputer programs to be written faster, more easily - and more economically.

The user does not have to worry about obsolescence. An 8-bit computer and a BRIDOS operating system offer the same data-processing capability as a 16-bit computer.

In networks and multi-user applications, data files can be shared quickly, simply and dynamically. In the 1.5 version of BRIDOS, as many as 16 users can access the files on a hard disk without loss of speed or efficiency. They can also overlap on the same file or files.

BRIDOS includes a complete access control system for protection of files. Each user has a private password and can easily designate the user or group of users who have access to his files. Access can be be either Read Only or Read/Write.

The printer spooler and de-spooler are designed for administration of printing queues based on user priorities and codes for paper formats. As many as 9 printers and 256 different format queues can be supported.

BRIDOS also includes an internal electronic mail system, which is designed primarily for system administration but can easily be employed for high-speed transfer of messages between users.

BRIDOS works with larger files and larger volumes than any other operating system for microcomputers. A volume or a file can be as large as 64 Mbytes.

BRIDOS uses memory space more efficiently. It can reside in a separate memory bank or a separate computer. The user has access to the same application program area as CP/M, with the additional benefit of index file support.

BRIDOS includes a number of utility programs. One of these can be used to convert CP/M file formats to BRISAM formats for faster loading and more efficient utilization of disk space. Files can also be accessed directly through BRISAM, for unmatched interactive file performance.

Simplified installation routines facilitate implementation of BRIDOS systems on various types of microcomputers.

Access control and allocation of disk space

Conventional multi-user operating systems for microcomputers divide a hard disk into physical volumes. A user logs in on a letter which represents a physical portion of the disk. The amount of physical space reserved for this letter is constant and does not necessarily correspond to the actual amount of data contained. In other words, the space remains physically allocated, whether it holds 4 K bytes of data or 4 M bytes.

One of the problems involved with physical allocation of disk space is that it reduces the speed with which users can access the files in a volume, since a file is tied to a physical location on the disk.

Another problem involves the use of programs which generate temporary files, such as WORDSTAR and COBOL compilers. The system must include a facility for ensuring that only one user at a time is processing a file in a given volume with such a program, and the procedures available for doing this are generally awkward, time-consuming and ineffective.

In contrast to conventional systems, BRIDOS allocates disk space in terms of logical volumes, not physical ones. A letter does not represent a physical portion of the disk. Instead of attaching a letter to a user, BRIDOS attaches letters to volumes, within which 7 levels of sub-volumes can be created. Any user can attach any letter to any volume to which he has access rights. The creator of a volume uses simple, straightforward routines to grant or deny access to other users, i.e. Read Only, Read/Write or No Access.

BRIDOS also allocates disk space dynamically. This means that the space occupied by a volume increases or decreases as data is entered or removed, so that the space always corresponds to the actual amount of data in the volume.

For example, after entering the system with his private password, User 1 attaches the letter "B" to DOCTXT and then logs in on "B". User 1 has previously created the volume DOCTXT. Within DOCTXT he has created 4 sub-volumes, DOCMAN, DOCCON, DOCLET and DOCPRO. These volumes contain the texts of manuals, contracts, letters and current projects, respectively.

User 1 has assigned access rights as follows: all users have access to DOCMAN. Users 2, 4 and 5 have Read-Only access to DOCCON and DOCLET; access to these files is denied to all other users (except 1, of course). Users 2 and 5 have Read/Write access to the files in the sub-volume DOCPRO.

As the files in DOCPRO may be processed with programs that generate temporary files, User 1 has also assigned exclusive access to DOCPRO. This means that only one user at a time has access to this volume. As soon as a user has finished working in the DOCPRO volume, any other authorized user has access to it automatically. Exclusive access can easily be assigned on either a temporary or a permanent basis.

When users other than User 1 enter the system with their passwords, they can also attach "B" to DOCTXT by simply writing ATTACH B:=DOCTXT. Access to the files in any volume is assigned by its creator, so that users logged into "B" have access to the files in the volume attached to "B" in accordance with the access rights assigned by User 1.

Allocating disk space logically and dynamically provides a number of major benefits.

First, utilization of disk space is highly optimized, as the physical space actually occupied by a volume is a function of the quantity of data it contains, not of an arbitrary - and more or less permanent - division of the disk.

Second, it takes much less time to locate a file in a volume.

Third, BRIDOS is more convenient than other multi-user systems. Any user can attach a letter to any volume, independent of the physical location of the volume on the disk. Access is defined by the creator of the volume.

Fourth, networking is greatly facilitated. A user can attach any local and/or any remote volume to any letter, i.e. he can attach a volume from his own local disk or from the central disk server.

Fifth, the entire access control system is user-oriented, highly flexible and easy to operate. The creator of a volume can assign access in terms of a specific user, a group of users or all users - without wasting time and without recreating files physically.

Sixth, BRIDOS provides a simple solution to the problem of processing a file with a program that writes temporary files. Exclusive access can be assigned very easily, and standard access codes are reactivated automatically as soon as a user has exited from a volume.

Typical network structure

A typical BRIDOS network with both central and local disk access is described below. It should be emphasized that this is only one of a virtually unlimited range of combinations.

This network is based on a central unit in which BRIDOS resides. The hardware in this unit consists of a CPU with a 64 KB RAM and a 10 MB Winchester drive. All data in the system is stored here. The unit also includes three processors for application programs run by users from terminals.

The central unit serves three microcomputers (with their own CPU's and diskette stations) and three dumb (as well as inexpensive) terminals. The microcomputers and terminals can be supplied by different manufacturers.

The system can be either interrupt-driven or polled, so that either synchronous or asynchronous hardware can be used for communication between an application processor and the file management center.

File management and the data base are completely separated from data processing. All file management within the system is handled by BRISAM, the file-management system incorporated in BRIDOS.

Two important points should be noted in this connection. First, BRIDOS is considerably faster than any other combination of operating and file-management systems available for microcomputer applications.

COBOL-80 (Version 4.01) running under BRIDOS has been benchmark-tested against COBOL-80 running under CP/M (Version 2.2), using identical program source codes. These tests clearly demonstrate the vastly superior performance of BRIDOS, which runs three (3) times as fast for sequential files and forty-seven (47) times as fast for index-sequential files. Test programs and source codes are specified in a booklet entitled "File Management Benchmarks" which describes the differences between CP/M and BRIDOS and is available from Bris Data.

The second point is that the quantity of information in the communications network is minimized, because the entire file-management system resides in the central unit. BRISAM is used for all file-management. In operation, the application program asks for information by means of a search key. All information is retrieved by BRISAM and then transmitted to the application-program processor, which serves either a terminal or a microcomputer.

In conventional systems, the application processor must be used for searches. It also has to handle information about the physical location of data on the disk. The processor searches and updates the index tree of the file-management system. The master processor in a conventional system handles safety, queuing and communications functions between storage media and application processors.

This approach has two major disadvantages:

1. Large quantities of information must be transmitted between the master processor and the application processors. This usually requires expensive hardware to ensure effective communications.

2. The conventional system is highly sensitive to a fault in an application processor. For example, if an index file is not updated, it will be difficult to retrieve the input data.

Faster, more efficient communication

BRIDOS works faster. Its operations are centralized. At any given moment, there is a smaller quantity of data in the communications network. Among other things, this means that a BRIDOS network can be connected to slower links such as modem/telephone lines and can utilize them with maximum efficiency.

The centralized file-management in the BRIDOS system reduces the demand for high-speed communication between processors. Cheaper hardware can be used - with the same, or even better system performance.

BRIDOS thus opens the door to on-line applications through the communications network.

Each work station in a BRIDOS network operates independently of the others. If one user wants a file in order to write a report, and another user wants to process the same file statistically, BRIDOS can serve them both without any noticeable delay.

Degradation close to zero

Degradation is the concept which describes the percentage deterioration in the operating efficiency of a system as each user is connected to the network.

In conventional networks and multi-user systems, a single processor is normally used for all file management as well as all data processing. The problem is obvious: operating speed decreases with the number of users.

The BRIDOS multi-user system avoids this sort of crippling effect. Each work station is served by its own processor. A user does not generate a load on the rest of the network until information is requested from the central register.

Very small quantities of information are transmitted in the BRIDOS network for search and updating of files, so that there is no measurable degradation for data-processing applications, no matter how many users are being served.

Maximum data security

Data security is just as important as speed and efficiency. BRIDOS comes as close to total data security as current software technology permits.

BRIDOS is designed for fast updating. The system ensures that each transaction is completed in its entirety. The risk of updating a data file without updating the index file is absolutely minimal. This is the best possible protection in the event of potential disasters such as power failure or unintentional reset.

Application software and file-management are physically separated. A fault in an application program can never crash information registers or interfere with the performance of other programs.

Simultaneous updating of index files and data files completely eliminates the risk of crashed files.

Technical characteristics of the BRISAM file-management system

BRISAM offers high-speed index sequential search in ascending or descending key order. A file can be as large as 64 MB. The number of records in a file is limited only by overall disk capacity.

A special buffering mode is used for transactions consisting of more than one operation. Transaction time is dramatically reduced.

State-of-the-art technology including a B+ tree and advanced buffer management generates extremely fast file-update operations. BRISAM has been tested against a typical file-management system for microcomputers. BRISAM executed INSERT, DELETE and UPDATE 5 times as fast. On search operations, BRISAM was 2-3 times as fast.

Files are fully updated at the end of each transaction. Other operating systems cannot execute updating unless the file has been closed. If BRISAM is running and a power failure occurs, the updated file is not lost. If the user happens to to hit the RESET button, data will not disappear.

A highly optimized cache-buffering system is used internally to minimize the number of disk accesses and guarantee data safety. A file operation that writes to the disk is first completed through the internal buffers. If the operation has to be aborted, nothing is written on the disk.

Disk space is allocated dynamically. A file can be as large as 64 MB or as small as 512 bytes. Disk space is automatically allocated or deallocated in pages of 512 or 1024 bytes as a file grows or shrinks. Pages are positioned as close as possible to the optimal position on the disk.

BRISAM ensures highly efficient utilization of disk space. An index sequential file built in random key order is packed to 85% of capacity. With ordered inserts - ascending or descending - the file is packed to 100% of capacity. Files do not have to be reorganized for packing.

When records are deleted from a file, disk space is reclaimed. Pages are always at least half-full.

Sequential files are also supported for data with fixed or variable record length.

Special program-load files allow the memory to be loaded at any address. For each file, one main and as many as 254 secondary entry points can be used.

A volume directory is maintained in sorted order. The directory is accessible by index sequential search operations, using the file name as a key.

For each volume, the name, date of creation and date of last backup are maintained. This information is presented on the screen as soon as a user starts work on a volume.

Volumes are organized as hierarchies. A volume may contain 7 levels of sub-volumes.

BRISAM includes functions for multi-key access as well as generation of unique keys.

1.1 INTRODUCTION

If you have a single-user BRIDOS Uni-system (see Section 1.01), you do not have to read Sections 1.2-15.10. All necessary information on BRIDOS utilities is provided in Section 16.1 and subsequent sections.

Users with Omni- and Multi-systems should read Sections 1.2-15.10 as well as 16.1 and subsequent sections.

In the following text, the command ^C indicates that the CONTROL key should be held down while the C key is pressed. <CR> = Carriage Return. Screen displays are printed in denser type, with messages from the BRIDOS system in lightface and user input in boldface, e.g.:

Enter user ID: **PDC**

Spaces have been inserted in the text of user commands for the sake of clarity alone. Thus **SYS <CR>** means "Enter SYS and then Carriage Return", not "Enter SYS and then space and then Carriage Return".

The word "terminal" refers to either a terminal or a satellite station which may or may not feature local disk access.

1.2 THE SYSTEM MANAGER (SYS)

After a BRIDOS multi-user system has been physically installed and initialized, a System Manager should be appointed. The System Manager is responsible for operation of the BRIDOS system and is the only user with the ID (user identification code) "SYS".

1.3 START OF SYSTEM

The system hardware should be started in accordance with the manufacturer's instructions. SYS should then run the Install Security program by entering **INSTSEC <CR>**. This program should be run only once and then erased from the A-drive. It should be stored on a floppy disk. On completion of INSTSEC, SYS enters:

LOGOFF <CR>
Enter user ID: **SYS <CR>**
No password exists for SYS. Run PASSWORD.

2.1 PASSWORD PROGRAM

This program is used to register passwords for all users, including SYS. To run the program, enter:

```
A>PASSWORD <CR>
Enter new password:
Enter new password again:
New password registered for SYS
A>
```

Registration of a password requires entering it twice (as an extra security measure), each time followed by <CR>. When a password is entered it is not displayed on the screen, which makes it more difficult for someone else to learn it. Thus in the above example SYS has entered a password twice and received confirmation that it has been registered. SYS can change this password at any time. SYS can remove a user password by running SYSACC (see Section 3.5).

SYS can assign a password to a user by logging on with a user ID and then running PASSWORD. The user can later run PASSWORD to change the password assigned by SYS. SYS has no way of knowing what the password is after the user has changed it. The only information available to SYS is that it has been changed, i.e. if SYS enters the user ID and the original password the system will reject the latter.

A password can contain a minimum of 4 and a maximum of 60 characters. The characters may be capital letters, small letters or control characters. However, the password character sequence must be entered in exactly the same form each time, i.e. if the password is POweR it must be entered as POweR, not as POWER.

2.2 PASSWORD CHECK

SYS should now check that the correct password is registered in the system. SYS enters:

```
A>LOGOFF <CR>
SYS logged off terminal N at xx:xx:xx
Enter user ID:
```

N is the number of the terminal and xx:xx:xx is the time of day (24-hour clock). SYS should now enters his/her user ID.

```
A>LOGOFF <CR>
SYS logged off terminal N at xx:xx:xx
Enter user ID: SYS <CR>
Enter your password:
```

SYS enters his/her password followed by <CR>.

SYS logged on to terminal N at xx:xx:xx

This shows that the correct password is registered.

3.1 REGISTERING USERS

SYS and only SYS can register users as individuals. Groups of users can be registered by SYS or other users. Both individuals and groups are registered by entering a 3-letter ID. Each ID - whether individual or group - refers to a unique user or group of users, i.e. two identical ID's cannot be entered for different users or groups of users.

3.2 REGISTERING INDIVIDUAL USERS

SYS can register other users by running the program SYSACC. This program can be run at any time, but only by SYS. To run the program, SYS enters:

```
A>SYSACC <CR>

 ^C   Return to system
 1    Admit new user to system
 2    Remove a password
 3    Deactivate a user
 4    Remove a deactivated user
 5    Display all users
 6    Directory map

 Choose function: 1 <CR>
 Enter user ID: PDC <CR>
 Enter default spooler priority: 55 <CR>
 PDC has been admitted as a registered user
 Enter user ID:
```

As the above example indicates, SYS enters a 3-letter user identification code for each user to be registered. This code can be entered in capital or small letters, or any combination of these, i.e. PdC or PDC. Unlike a password, it does not have to be entered in the same form each time. The code is stored and displayed in capital letters.

The program continues to ask for user ID's until SYS responds with <CR> and returns to the main SYSACC menu.

The default spooler priority refers to the position in a printer-spooler queue which will automatically be assigned to PDC whenever he enters the system (see Section 15.1). SYS may enter any number between 0 and 255. If SYS enters <CR> instead of a number, a user priority of 100 is automatically assigned. Priority is defined in inverse numerical order, i.e. the lower of two numbers has higher priority.

Each printer queue corresponds to a paper format. Within the queue, files are ordered for printing in accordance with user priorities.

If files are being sent to a printer queue simultaneously by PDC and by another user with a lower priority number, the other user's files will be printed first. The default priority number can be overridden when a file is submitted to the printer queue, i.e. a higher or a lower priority can be assigned.

3.3 REMOVE A PASSWORD

Although a user password is known only to the user, SYS can remove a password from the system by selecting Function 2 in the SYSACC menu. When the user attempts to log on, the system announces that no password exists and that PASSWORD must be run. This function is a safeguard for a user who cannot remember his/her password.

3.4 DEACTIVATE A USER

Function 3 in the SYSACC menu enables SYS to deactivate a user. A deactivated user can no longer log on to the system and is automatically removed from all groups of which he/she is a member. SYS automatically becomes the owner of all groups owned by the user.

A user should remain deactivated until all the volumes that he/she owns have been reassigned or erased by SYS, who may then remove the user from the system.

3.5 REMOVE A DEACTIVATED USER

Function 4 in the SYSACC menu enables SYS to remove a user. This function should not be used until a) a user has been deactivated and b) SYS has either reassigned or erased all the volumes owned by the user.

3.6 DISPLAY ALL USERS

Function 5 in the SYSACC menu enables SYS to obtain a list (display) of all registered users and their printer-queue priorities. If a printout of this list is required, enter ^P before entering 5 (see Section 15.2).

3.7 DIRECTORY MAP

Function 6 in the SYSACC menu displays a) all the volumes on a physical drive, b) owners and c) access rights, including exclusive-use attributes. If a user or a group has been deactivated or removed, the user or group ID will be displayed in parentheses.

SYS can use this function to become the owner of volumes whose owners have been deactivated or removed from the system.

When SYS automatically assumes ownership of a volume whose owner has been deactivated/removed, the existing access rights to the volume are maintained.

When a user (or group) is deactivated by SYS, all access rights available to the user (or group) are automatically suspended.

3.8 REGISTERING GROUPS OF USERS

Any user (including SYS) can register a group of users by running the program GROUPS. This program is also used to change, deactivate and display groups of users.

Each group is owned by the user who creates it. The owner automatically becomes a member of the group. The owner and the owner alone can remove/add members or deactivate a group. When a group is deactivated, SYS automatically becomes the owner and sole member.

A group may contain an unlimited number of members. A group may not become a member of another group; only individual users can be members of a group.

SYS and only SYS can display all groups in the system (see Section 3.6).

Groups of users can be registered by entering **GROUPS <CR>**.

3.9 NEW GROUPS

```
+A> GROUPS <CR>

1   Display all groups owned by PDC
2   Display all groups in which PDC is a member
3   Create/deactivate/change a group

Choose function: 3 <CR>

Enter group ID: GRO <CR>

This is a new group

Group: GRO  Owner: PDC
  PDC

1   Add members to group
2   Delete members from group
3   Store group and return to main menu
4   Return to main menu without changing group

Choose function: 1 <CR>

Enter new member: LGB <CR>

Group: GRO   Owner: PDC
  PDC  LGB
Enter new member:
```

In the above example, PDC enters the name of a group which is not in existence. The system announces that GRO is a new group and then lists a) the name of the group, b) the owner and c) the members. The creator of a group automatically becomes a member of the group. Obviously, the only initial member is PDC, who is listed below the text Group: GRO, owner: PDC. Enter <CR> to return to the main menu.

The + before the A> indicates that PDC has entered ^P and that the data on the screen is thus also being sent to a printer or to a file (see Section 15.2).

3.10 EXISTING GROUPS

+A> GROUPS

1 Display all groups owned by PDC
2 Display all groups in which PDC is a member
3 Create/deactivate/change a group

Choose function: 3 <CR>

Enter group ID: TOP <CR>

This is an existing group

1 Change this group
2 Deactivate this group
3 Choose another group

Choose function:

Compare the above menu with the one for a new group in Section 3.9.

3.11 ADD MEMBERS TO GROUP

To add members to a new group, see Section 3.9. To add members to an existing group:

1 Display all groups owned by PDC
2 Display all groups in which PDC is a member
3 Create/deactivate/change a group

Choose function: 3 <CR>

Enter group ID: TOP <CR>

This is an existing group

1 Change this group
2 Deactivate this group
3 Choose another group

Choose function: 1 <CR>

1 Add members to group
2 Delete members from group
3 Store group and return to main menu
4 Return to main menu without changing group

Choose function:

Note that each time PDC adds a member to the group the system responds by listing the name of the group, the owner and - on the next line - the members in the order they were added, e.g.:

```
Group: TOP  Owner: PDC
  PDC GHB DTH AMS LGB
Enter new member:
```

If no more members are to be added, enter <CR> to return to the main GROUPS menu.

3.12 DEACTIVATE A GROUP

```
1   Display all groups owned by PDC
2   Display all groups in which PDC is a member
3   Create/deactivate/change a group

Choose function: 3 <CR>

Enter group ID: TOP <CR>

This is an existing group

1 Change this group
2 Deactivate this group
3 Choose another group

Choose function:2 <CR>

Do you really want to deactivate this group? (Y/N) Y <CR>

1   Display all groups owned by PDC
2   Display all groups in which PDC is a member
3   Create/deactivate/change a group

Choose function:
```

When a group is deactivated, SYS automatically becomes the owner and sole member. The group name remains reserved, i.e. it cannot be assigned to another group (or individual) until SYS removes the group from the system. SYS removes the group by a) running SYSACC function 6 (see 3.7) and removing any access rights that were assigned to the group b) running GROUPS and then selecting function 3 above.

3.13 DISPLAY ALL GROUPS

When the GROUPS program is run by SYS, a special menu is displayed:

```
+A> GROUPS

1  Display all groups owned by PDC
2  Display all groups in which PDC is a member
3  Create/deactivate/change a group
4  Display all groups
```

Note that Function 4 is not available when GROUPS is run by anyone but SYS.

3.14 INFORMATION TO USERS: ID AND PASSWORD

When SYS has finished registering users, each user should be notified of his/her ID as soon as possible. Each user should then log on to the system as soon as possible and enter his/her user password by running the program PASSWORD (see Section 2.1).

4.1 LOGGING ON

A user logs on to the system by entering an ID. As usual, the system then asks for a password. If the user does not have a password, he/she is instructed to run PASSWORD (see Section 2.1). PASSWORD should also be run if the user has a password assigned by SYS and wants to change it, or if there is reason to believe that someone else knows the password. Screen display:

```
Enter user code: PDC <CR>
Enter password:

PDC logged on to terminal 01 at 10:13:26

The I/O byte assignments are:CON:=TTY:RDR:=TTY: PUN:=TTY: LST:=CRT:

The default spooler queue number is 0
```

The I/O byte assignments are specific to the system hardware. Consult your hardware supplier for details.

For an explanation of the default spooler queue number, see Section 15.1.

A user can always change the default spooler queue and set an automatic LOGON command by running CONFIGU (see Section 6.4).

5.1 PRINTOUTS WITH ^P

As in CP/M, the command ^P is used to obtain printouts of user input and screen displays. This command is a switch, i.e. entering it turns the printing facility on. Entering it again turns the facility off.

In contrast to CP/M, BRIDOS indicates whether the printout function is on by listing a plus-sign before the prompt:

```
+A>
```

In a multi-user system, entering the ^P command sends screen displays to the list device. If the list device is the BRIDOS spooler, the displays will be stored on the disk as a file. This file can then be put into the spooler queue (see Section 15.2). Consult your hardware supplier for list-device assignments.

6.1 CONFIG: SETTING UP THE COLD-START ENVIRONMENT

Two CONFIG programs are included in BRIDOS: CONFIGT and CONFIGU.

CONFIGT is used to assign an LST (list) device for a terminal.

CONFIGU is used to set a) the user's default log-on command and b) the default paper format (see Section 15.1) in the printer-spooler queue.

6.2 CONFIGT

CONFIGT is used to identify the type of device to which data is sent for listing when the terminal is in operation. This device is normally a printer. The terminal can send data either through the central BRIDOS printer-spooler or through a local port.

The assignment of default LST devices for a terminal follows the CP/M conventions for the I/O byte, i.e. the LST or list function can be assigned with the following logical device codes:

TTY: teletypewriter
CRT: cathode ray tube
LPT: line printer
UL1: user list device

Note that for most modern microcomputers the names of list devices are logical concepts and do not refer to specific physical units. Consult your hardware supplier for detailed information.

To run the program, the user should be logged on to A:.

```
+A>CONFIGT <CR>
Bootable volume on default drive. Permanent assignments can be made.

1: Set I/O byte (CON:=TTY: RDR:=TTY: PUN:=TTY: LST:=CRT:)
4: Make permanent assignments on disk and exit to BRIDOS

Choose function:
```

Function 2 is not normally available. It is used to set up a cold-start command for the terminal and is available only if the access control system has not been installed (as on single-user floppy-disk systems). Access control for multi-user systems is usually installed by the hardware supplier. Function 3 is reserved for hardware initialization and will be available in a future release of BRIDOS.

```
+A>Choose function: 1

The current assignments are:

CON:=TTY:       TTY: CRT: BAT: UC1:
RDR:=TTY:       TTY: PTR: UR1: UR2:
PUN:=TTY:       TTY: PTP: UP1: UP2:
LST:=CRT:       TTY: CRT: LPT: UL1:

*RDR:=UR1: <CR>

CON:=TTY:       TTY: CRT: BAT: UC1:
RDR:=UR1:       TTY: PTR: UR1: UR2:
PUN:=TTY:       TTY: PTP: UP1: UP2:
LST:=CRT:       TTY: CRT: LPT: UL1:

*<CR>

     1: Set I/O byte (CON:=TTY: RDR:=UR1: PUN:=TTY: LST:=CRT:)
     4: Make permanent assignments on disk and exit to BRIDOS

     Choose function:
```

The available assignments are listed to the right of the current ones. As each device is assigned, a new list of current assignments is displayed. When no more assignments are to be made, enter <CR> and then select Function 4.

To change the default LST device, run CONFIGT again.

6.3 TEMPORARY LST ASSIGNMENTS

Input data for CONFIGT are stored in the boot files for terminals. These files are located on physical drive 1: and are named SLAVEXX.SYS, where XX is the terminal number (2 digits, e.g. 01). Boot files can be write-protected by SYS (see Section 11.1) but by no one else. Boot files should be write-protected in order to avoid the confusion that would result if all users could make permanent changes in the cold-start environment.

If the boot files are write-protected, a user can make temporary assignments by a) running CONFIGT, making assignments with Function 1 and then entering ^C to return to BRIDOS without selecting Function 4, or b) entering CONFIGT and the assignments on a single line, e.g.:

```
+A>CONFIGT RDR:=UR2: PUN:=UP1:

     The current assignments are CON:=TTY: RDR:=UR2: PUN:=UP1: LST:=CRT:
```

The cold-start environment will revert to the assignments in the boot files as soon as the user has logged off.

6.4 CONFIGU

CONFIGU is used to:

1) Set up a cold-start routine, i.e. a sequence of commands which is initiated automatically when a user logs on.
2) Set up a user-specific default paper format for the printer-spooler.

In many applications, a user processes files in a BRIDOS volume with the same application program every time he/she logs on. Setting up a cold-start command enables the user to go straight to work without having to enter any commands other than an ID and a password. Logging on activates a command file which literally replaces the user by feeding information to the computer.

For example, let us say that PDC is in the habit of using WordStar® to work on files in a volume called HELLO. He frequently calls in other programs, such as the The Word Plus®, through the "Run a program" function in WordStar's main menu.

First, PDC uses WordStar (in non-document mode) to write a file called STARTPDC.SUB. This file consists of the following lines of text:

```
ATTACH B:=HELLO
B:
WS
LOGOFF
```

PDC then runs CONFIGU by entering:

```
+A>CONFIGU <CR>

1: Set default spooler queue (   0)
2: Set logon command (                                    )
3: Make permanent and return to system

Choose function: 1 <CR>
Enter default spooler queue number (0...255): 20 <CR>

1: Set default spooler queue ( 20)
2: Set logon command (                                    )
3: Make permanent and return to system

Choose function: 2 <CR>
Enter logon command (Space for none): SUBMIT STARTPDC <CR>

1: Set default spooler queue ( 20)
2: Set logon command (SUBMIT STARTPDC                     )
3: Make permanent and return to system
Choose function: 3 <CR>
```

The log-on command may consist of a maximum of 32 characters. It may also consist of a simple instruction to call in a program, e.g. WS.

The next time PDC uses the system he enters his ID and password. BRIDOS then goes to the file STARTPDC.SUB and reads this file as instructions to a) attach B: to the volume HELLO, b) log in on B: and c) call in WordStar.

When PDC has finished working, he gives the "Exit to system" command in WordStar's main menu. STARTPDC.SUB then takes him straight out of the BRIDOS system, i.e. it logs off.

Note that this log-on command is specific to the user, not to the terminal. It will be initiated at any terminal as soon as PDC logs on to the system.

In the above example, PDC specified a default spooler-queue number of 20 (see Section 15.1). This code refers to a paper format. PDC can thus avoid specifying this format each time he submits a file to the printer-spooler.

The default paper format can be overridden when a file is submitted to the printer-spooler (see Section 15.4).

7.1 PHYSICAL AND LOGICAL VOLUMES

In BRIDOS, a physical volume consists of all the data on a physical medium. All the files on 1: thus constitute the physical volume on drive 1:. A physical medium is referred to by a number and a colon. For example, in a computer system with a hard disk (e.g. a Winchester drive) and a single floppy-disk backup, the hard disk may be designated 1: and the floppy disk 7:. Numbers 2-6 may be reserved for disk cartridges or for daisy-chained hard disks. The code 7: refers to the floppy-disk drive, irrespective of which floppy disk is actually present in the drive.

A logical volume is a concept. It includes a group of files and an index, which is called a directory. These files can be located anywhere on the physical drive. File size is limited only by disk capacity, and a single file may be as large as 64 million bytes.

Logical volumes in BRIDOS are organized as hierarchies (resembling the structure in another operating system called UNIX). At the top of the hierarchy is the root volume on a given physical medium. All other logical volumes on that medium are sub-volumes of the root volume. The BRIDOS hierarchy is 7 levels deep.

A user creates a logical volume with a program called VOLUME (see Section 8.1). The user who creates a volume becomes its "owner" and is the only person who can assign access rights to it.

The name of a logical volume consists of 1-8 characters (alphabetic or numeric). The directory of a volume is a file. It is named with the volume name and the extension .DIR. A volume named HELLO thus has a directory named HELLO.DIR.

A logical volume cannot be used until a letter is attached to it. For letters B-H, the command ATTACH is used. The letters are local for each terminal, so that any user at any terminal is free to attach them. After a letter has been attached to a volume, a reference to the letter is a reference to the logical volume and thus to its directory and the files it contains. These attachments are transient, i.e. they are discontinued by the command DETACH or as soon as the user logs off.

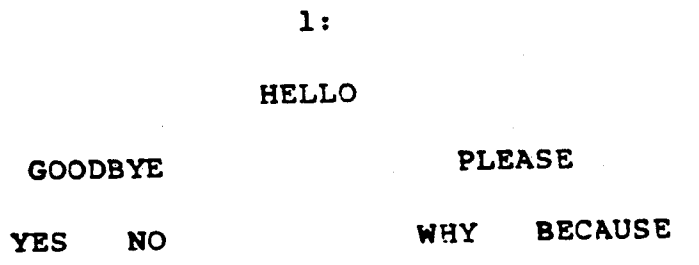
There is one exception to this rule. The letter A is permanently attached to the root volume on the first physical drive. Thus in the system referred to above, A: is permanently attached to 1:. The root volume is indexed by the root directory.

In a single-user system with floppy disks only, each physical diskette contains a root volume only. There are no sub-volumes. A removable physical medium cannot have sub-volumes.

Directories of the sub-volumes one level down from the root volume are listed as files with the extension .DIR. A .DIR file is thus a directory of other files one level below.

As attachments with all letters other than A: are transient, a user who logs on to the BRIDOS system finds that all the files in the root volume are attached to A:. Some of these files have the extension .DIR and are directories of sub-volumes one level below the root volume. All users have access to all the non-directory files in the root volume. A user has access to directory files if he/she has access to the sub-volumes containing them.

A user is thus free to attach any letter (B-H) to any of the volumes on the first physical drive, provided that he/she has access to them. The diagram below illustrates the general structure of the system.



Neither the root volume on 1: nor its directory has a name, as they are both attached permanently to A:. As mentioned above, a reference to a letter that is attached to a volume is also a reference to the volume. A reference to A: is therefore always a reference to the root volume and the root directory.

In the diagram above, HELLO is a sub-volume of the root volume (one of many possible sub-volumes, their number limited only by disk space). The file HELLO.DIR is the directory of HELLO. The file GOODBYE.DIR is the directory of the volume GOODBYE, which is a sub-volume of HELLO. The files YES.DIR and NO.DIR are directories of the volumes YES and NO, which are sub-volumes of GOODBYE. The volume PLEASE is another sub-volume of HELLO. The file PLEASE.DIR is the directory of PLEASE. The files WHY.DIR and BECAUSE.DIR are directories of two sub-volumes of PLEASE. BRIDOS permits seven hierarchical levels of sub-volumes.

Naturally, the root volume on physical drive 1: could contain many different files besides directories for sub-volumes, such as HELLO.DIR. These files may have extensions such as .COM, .TXT, etc.

A user cannot attach a letter to a sub-volume (such as NO) without having previously attached to the volumes above it in the hierarchy, i.e. first to HELLO and then to GOODBYE. If a user attempts to attach a letter to a sub-volume without having first attached to the volume(s) above it, the system displays an error message. This means that a user can only attach a letter to a volume for which a .DIR file can be listed on the screen.

Thus if the user is unaware of the hierarchy of volumes, he/she will be automatically directed downwards by the system in accordance with the hierarchy. An individual user can attach any letter to any volume, but only one letter at a time can be attached to the same volume by the same user.

To repeat, a volume is a concept (or envelope) which includes files and a directory. A volume at a given level is indexed by a directory at the same level. Thus, the volume NO is indexed by the directory file NO.DIR.

7.2 FILE NAMES

File names follow the CP/M conventions, i.e. a name can consist of up to 8 characters (alphabetic or numeric) with an optional extension of a dot and up to 3 characters (alphabetic or numeric). Example: START.PDC.

Two or more files with the same name can exist on the same physical disk, but not within the same logical volume.

7.3 PROGRAM FILES

The files that constitute a program are usually stored in the root volume, to which A: is permanently attached. All users have access to files in the root volume. BRIDOS enables all users with access to a file to get at it simultaneously. There is thus no reason to make copies of program files.

SYS can deny a user access to a program by taking it out of the root volume, inserting it into a sub-volume and blocking access.

SYS normally write-protects all program files so that other users cannot remove them, change them or destroy them (see Section 11.1).

When a user calls in a program, BRIDOS immediately searches the volume attached to the logged letter, i.e. if PDC has attached B: to HELLO and then logged in on B:, BRIDOS searches HELLO for the program. If BRIDOS doesn't find the program in HELLO, it goes directly to A: and looks for it there. BRIDOS searches at such high speeds that a user doesn't usually notice the delay, no matter how many terminals are in use. If a disk-intensive program is being run, there may be some delay, depending on the type of program, the type of disk controller and the number of terminals in use.

7.4 PROTECTING FILES

When a file is write-protected, no one can make any changes in it. A user can write-protect the files in a volume he/she creates, so that access to a volume can be given to other users while specific files in the volume can be protected. Only SYS can write-protect the files in the root volume on a non-removable physical drive.

Two utility programs called PROTECT and ENABLE are used to assign and remove the write-protect attribute (see Section 11.1).

7.5 ATTACHMENT

A user normally attaches a letter to a volume with the ATTACH utility and then logs on to the letter. This should not be confused with logging on to the system (ID + password). For example, after PDC has logged on to the system, he attaches the letter C: to the volume HELLO and then enters C: <CR> to log on to C:.

Any authorized user can attach a letter to a volume, but only one letter at a time can be attached to the same volume at the same time by the same user. Thus PDC, GHB and DTC can all attach C: to HELLO.DIR from different terminals.

7.6 OWNER PRIVILEGES

The creator of a volume "owns" it and is the only person who can grant access to other users. An unauthorized user who attempts to attach a letter to a volume is informed that he/she does not have access. NOTE: If a user is deactivated, SYS can take over ownership of the volumes created by that user (see Section 3.4).

The owner of a volume has two other special privileges. The owner is the only one who can:

1) Create sub-volumes to the volume
2) Change the write-protect attribute for a file in the volume.

Note once again that SYS alone can change the write-protect attributes for files in the root-volume on a non-removable physical drive.

7.7 DIRECTORY LISTINGS

The files on the root volume have the usual CP/M file names, consisting of an element of up to 8 characters (alphabetic or numeric) and an optional extension. When a user creates a volume, he/she enters only the first element of the name, e.g. HELLO. The system adds the extension .DIR, which shows that HELLO.DIR is the directory of the volume HELLO. A directory listing of the root volume may show files which end in .COM, .DOC, .LET, .DIR, etc. Extensions other than .DIR are added by users or by programs.

We can also note once again that HELLO.DIR is actually the directory of the sub-volume HELLO of the root volume on drive 1:. If a volume is created within HELLO, it will be a sub-volume of HELLO. A directory listing may look like this (PDC has attached B: to HELLO, C: to GOODBYE and then logged on to C:):

```
C>DIR <CR>

Directory for volume: C:=1:/HELLO.DIR/GOODBYE.DIR

SIMPLE.TXT EASY.CON NEVER.LET

2472 free pages, 10508 total pages, page length = 1024 bytes
```

This indicates that C has been attached to the volume GOODBYE, a sub-volume of HELLO. The volume GOODBYE has been created within HELLO. The volume GOODBYE contains three files, SIMPLE.TXT, EASY.CON and NEVER.LET. Of the 10508 pages on physical drive 1:, 2472 pages are free. Each page contains 1024 bytes.

You will not be surprised to learn that GOODBYE can contain files with the extension .DIR, and that these would be directories of sub-volumes of GOODBYE. BRIDOS allows up to 7 levels of sub-volumes, starting from A.

Let us repeat that a logical volume consists of a group of files and an index, or directory. A sub-volume thus consists of a group of files and a directory which is at the same level as the sub-volume.

7.8 CREATING A VOLUME

A volume can be created (see Section 8.1) by any user who is logged on. The creator of a volume automatically becomes its owner. The owner alone can assign or modify access rights or create a sub-volume (see Section 8.1).

If SYS does not own a volume, he/she cannot assign/modify access rights, change ownership or create a sub-volume.

8.1 VOLUME PROGRAM

To create a volume, PDC runs the VOLUME program by entering VOLUME.

```
+A>VOLUME

Enter volume name: HELLO <CR>

New volume

A:HELLO.DIR owned by PDC
No one has read/write access rights
No one has read/only access rights
This is not an exclusive use volume

^C   Return to system without creating volume
1    Change read/write access rights
2    Change read/only access rights
3    Change exclusive use attribute
4    Create this volume

Choose function:
```

The following points should be noted: A volume name can consist of 1-8 characters, alphabetic or numeric. The extension .DIR is added automatically by the system. The volume is provisionally created on the root volume, on physical drive 1:, to which A: is permanently attached. HELLO.DIR is not finally created until Function 4 has been selected.

The message "A:HELLO.DIR owned by PDC" means that PDC owns a volume called HELLO, which has a directory called HELLO.DIR.

8.2 ACCESS RIGHTS

When the volume is created, no one except the owner has access to it. Access rights can be assigned only by the owner. A user with access to a volume has access to all the files within the volume. Access rights can be:

1) Read/write: All files within a volume can be read and/or edited.
2) Read/only: All files within a volume can be read but not edited, i.e. no new files can be created, existing files cannot be changed.
3) Exclusive use: the volume can be attached by only one authorized user at a time. When an authorized user attaches the volume, all access rights are suspended, i.e. no other authorized user can attach it. Access rights are reactivated as soon as the volume is detached.

The VOLUME program can also be used to:

1) Change access rights to an existing volume
2) Erase a volume
3) Reassign ownership.

8.3 READ/WRITE ACCESS

```
A:HELLO.DIR owned by PDC
No one has read/write access rights
No one has read/only access rights
This is not an exclusive use volume

^C   Return to system without creating volume
1    Change read/write access rights
2    Change read/only access rights
3    Change exclusive use attribute
4    Create this volume

Choose function: 1 <CR>
Enter spaces, asterisks, user ID or group ID: GHB <CR>
```

Entering 3 spaces with the space bar means that no one except PDC has read/write access. Entering 3 asterisks means that all registered users in the system have access. Entering an individual ID assigns access to the holder of the ID. Entering a group ID assigns access to all the members of the group. In the above example, read/write access has been assigned to GHB.

8.4 READ/ONLY ACCESS

```
A:HELLO.DIR owned by PDC
GHB has read/write access rights
No one has read/only access rights
This is not an exclusive use volume

^C   Return to system without creating volume
1    Change read/write access rights
2    Change read/only access rights
3    Change exclusive use attribute
4    Create this volume

Choose function: 2 <CR>
Enter spaces, asterisks, user ID or group ID: *** <CR>
```

Entering 3 spaces with the space bar means that no one has read/only access. Entering 3 asterisks means that all registered users in the system have access. Entering an individual ID assigns access to the holder of the ID. Entering a group ID assigns access to all the (active) members of the group. In the above example, read/only access has been assigned to all registered users in the system.

8.5 EXCLUSIVE USE

```
A:HELLO.DIR owned by PDC
GHB has read/write access rights
*** has read/only access rights
This is not an exclusive use volume

^C   Return to system without creating volume
1    Change read/write access rights
2    Change read/only access rights
3    Change exclusive use attribute
4    Create this volume

Choose function: 3 <CR>

A:HELLO.DIR owned by PDC
GHB has read/write access rights
*** has read/only access rights
This is an exclusive use volume
```

Function 3 operates as a toggle. If a volume is not marked for exclusive use, selecting Function 3 assigns exclusive use. If a volume is marked for exclusive use, selecting Function 3 removes the exclusive-use attribute.

8.6 FINAL CREATION OF A VOLUME

```
A:HELLO.DIR owned by PDC
GHB has read/write access rights
*** has read/only access rights
This is an exclusive use volume

^C   Return to system without creating volume
1    Change read/write access rights
2    Change read/only access rights
3    Change exclusive use attribute
4    Create this volume

Choose function: 4 <CR>
A:HELLO.DIR successfully created

Enter volume name:
```

In the above example, PDC has selected Function 4. The system announces that the volume is now in existence on root volume A: and then asks for the name of another volume.

The process described above will be repeated if the name of a new volume is entered. Enter ^C to return to the system.

8.7 CHANGING ACCESS RIGHTS TO AN EXISTING VOLUME

The menu for changing the access rights to an existing volume differs slightly from the menus shown above. A user can change the access rights to a volume - **if he/she owns the volume** - by running VOLUME, entering the volume name and selecting the appropriate function(s). If a user attempts to change the access rights to a volume of which he/she is not the owner, the system will display a message such as:

```
This volume is owned by GHB. No one else can change access rights.
```

If PDC wants to change the access rights to HELLO.DIR, he enters VOLUME <CR>.

```
Enter volume name: HELLO <CR>

A:HELLO.DIR owned by PDC
GHB has read/write access rights
*** has read/only access rights
This is an exclusive use volume

^C   Return to system leaving original access rights unchanged
1    Change read/write access rights
2    Change read/only access rights
3    Change exclusive use attribute
4    Store new access rights
5    Erase this volume
6    Reassign ownership for this volume

Select function:
```

To change access rights, select Functions 1-3, enter the appropriate codes (see Sections 8.1-5) and then select Function 4.

8.8 ERASING A VOLUME

A volume cannot be erased until all the files in the volume have been erased with the ERA utility. Enter ERA followed by the name of the file. Wild cards may be used, e.g. *.TXT or SIMPLE.*. The system will ask whether a file is to be erased. A word-processing program can also be used to erase files.

To erase a volume, select Function 5.

```
A:HELLO.DIR owned by PDC
GHB has read/write access rights
*** has read/only access rights
This is an exclusive use volume

^C Return to system leaving original access rights unchanged
1    Change read/write access rights
2    Change read/only access rights
3    Change exclusive use attribute
4    Store new access rights
5    Erase this volume
6    Reassign ownership for this volume

Select function:5 <CR>
A:HELLO.DIR erased
```

8.9 REASSIGNING OWNERSHIP OF A VOLUME

The owner alone can reassign ownership of a volume. If an owner has been deactivated or removed from the system, SYS may reassign ownership by first becoming the owner (see 3.7) and then running VOLUME. The owner of a volume can reassign ownership as follows:

```
A:HELLO.DIR owned by PDC
GHB has read/write access rights
*** has read/only access rights
This is an exclusive use volume

^C Return to system leaving original access rights unchanged
1    Change read/write access rights
2    Change read/only access rights
3    Change exclusive use attribute
4    Store new access rights
5    Erase this volume
6    Reassign ownership

Select function: 6 <CR>
Enter user ID: GHB <CR>
GHB is now the owner of A:HELLO.DIR
```

9.1 ATTACH COMMAND

A volume cannot be used until a letter has been attached to it. The ATTACH command enables a user to attach any letter to any volume to which the user has access rights. The volume is specified by the first element of its name, i.e. without .DIR.

```
+A>ATTACH B:=HELLO <CR>

A:HELLO ATTACHED TO B:
```

In contrast to CP/M, all BRIDOS utilities display messages indicating that they have accomplished their tasks. The message above shows that the letter B: has been successfully attached to PETER.DIR, which is a sub-volume of the root volume, which is attached to A:.

If a user attempts to attach a letter to a volume to which he/she has no access, an error message is displayed:

```
You do not have access to this volume
```

9.2 CHECKING ATTACHMENTS

A listing of attachments (letters to volumes) can be obtained by entering:

```
+A>ATTACH <CR>

A:=1:
B:=1:/HELLO.DIR
```

This message indicates that B is attached to the volume HELLO.DIR, which lies on physical drive 1:.

As in CP/M, PDC can now log on to B: by entering:

```
+A>B: <CR>

Winchester                820922  820930
```

This message indicates the name of the physical drive on which the volumes attached to B: are located.

The first date (820922) is the date of creation of the root volume on the physical drive. The second date is the date of the most recent backup copy.

10.1 DETACH COMMAND

To detach a letter from a volume (e.g. F: from GOODBYE.DIR) enter:

```
A>DETACH F: <CR>

F: DETACHED
```

To detach all letters (except A:), enter:

```
A>DETACH *: <CR>
*: DETACHED
```

Once again, note that a BRIDOS utility always informs you that it has (or has not, and why) performed the task you gave it.

11.1 WRITE-PROTECT

A write-protected file cannot be changed or erased. Two utilities called PROTECT and ENABLE are used to assign and remove the write-protect attribute to a file. The name of the appropriate utility is entered, followed by the letter to which the volume containing the file is attached, and the name of the file:

```
A>PROTECT C:NEVER.LET <CR>
```

or

```
A>ENABLE C:NEVER.LET <CR>
```

Wild cards can be used, e.g. *.LET or NEVER.*. If the volume containing NEVER.LET is attached to the letter to which the user is logged on, it is not necessary to enter the letter, e.g.:

```
C>PROTECT NEVER.LET <CR>
```

12.1 DIR COMMAND

Enter the DIR command to display a directory of the volume to which the logged letter is attached. Assume that PDC has attached F: to GOODBYE and then logged on to F:.

```
+F>DIR <CR>

 Directory for volume: F:=1/HELLO.DIR/GOODBYE.DIR

 SIMPLE.TXT NEVER.LET

 2389 free pages, 10608 total pages, page length = 1024 bytes
```

A directory listing for a volume cannot be obtained unless a letter has been attached to the volume.

13.1 LISTING OF CURRENT USERS IN SYSTEM

A listing of users currently logged on to the system can be obtained by running the program USERS.

```
A>USERS <CR>

Terminal    User    Logged on at

  1         PDC     09:05:00
  4         GHB     11:23:16
```

14.1 COPYING FILES

Files can be copied from volume to volume or from physical drive to physical drive with the utility program PIP. Remember that a letter must be attached to a volume before PIP can be used.

Enter PIP, followed by the destination letter, a "=" sign, the source letter and the file to be moved.

PIP can be used to move files on to the Winchester drive. If the hard disk in a system is physical drive 1: and the floppy disk drive is 7:, the file WSMSGS.COM can be copied on to the hard disk from floppy disks by attaching F: to 7: and then entering :

```
A>PIP A:=F:WSMSGS.COM <CR>
```

To copy all the files from the floppy disk, enter:

```
A>PIP A:=F: <CR>
```

To copy only files WS.COM AND WSMSGS.COM, enter:

```
A> PIP <CR>
 *A:=F:WS.COM <CR>
 *A:=F:WSMSGS.COM <CR>
```

The asterisk and the question mark can be used as general indicators (wild cards) in PIP. To copy all the files with the extension .COM, enter:

```
A>PIP A:=F:*.COM <CR>
```

To copy all files with the name DATA and any extension, enter:

```
A>PIP A:=F:DATA.* <CR>
```

Normally there is no need to waste space by copying files from one volume to another. In contrast to CP/M, BRIDOS allows any user to access a file simply by attaching a letter to the volume containing it (assuming he/she has access rights to the volume).

If the message "Command syntax error" is displayed, the PIP command string has been improperly entered.

15.1 THE PRINTER-SPOOLER

The BRIDOS printer spooler is a complete facility for spooling and de-spooling files in a multi-user system. This means that the operator can submit files for printing and have them automatically inserted in a queue. The de-spooler selects the files from the queue and orders printouts.

Each queue corresponds to a paper format. Queue numbers range from 0 to 255. The default queue number (set with CONFIGU program) can be overridden when the file is submitted to the spooler. See Appendix 1 for a typical list of paper formats.

Within each queue, files sent to the spooler by users are selected for printing in accordance with the user's default spooler priority (see Section 3.2). The default user priority can also be overridden when the file is submitted to the spooler.

In general, the files submitted to a printing spooler are temporary files, i.e. they consist of data that are to be printed but not stored. The print file is therefore erased after printout. If the data are to be saved, an option must be specified when the file is sent to the queue (see Section 15.4).

This means that a source file such as a text file should normally not be sent directly to the spooler. Text files produced with WordStar (or any other word-processing program) can be submitted to the spooler through the WordStar print menu under the following conditions:

The CONFIGT program should be run to set the I/O byte for the terminal so that output to the CP/M list device will be sent to the spooler. If WordStar is installed to send to the CP/M list device, WordStar printouts will automatically be submitted to the spooler without any risk of erasing the text file.

To print and save a source file that cannot be sent through the WordStar print menu, specify the L-option when you send the file to the spooler - otherwise the file will be erased after printout (see Section 15.4).

The combination of user priorities and format codes enables printers to be utilized with maximum efficiency, so that the total investment in printers can be minimized.

The BRIDOS spooler can support as many as 9 printers and 256 queues. A printer requires one run-unit position in the central disk processor, so that a system with 16 run units can consist of 15 satellite processors and 1 spooling printer, 14 satellite processors and 2 spooling printers, etc.

15.2 ^P FILES

If the I/O byte has been set so that output to the LST: device goes into a spool file, entering ^P before console input will ensure that output data are sent to a spool file. Consult your hardware supplier about setting the I/O byte.

The contents of the spool file can be printed by entering:

```
F>SPOOL LST: <CR>
```

15.3 SPOOLER UTILITIES

The BRIDOS spooler includes 3 utility programs.

SPOOL is used to print ^P-files (see Section 15.2) and to:

a) Insert a file in a spool queue
b) Display a listing of files in the spool queue
c) Remove a file from the spool queue.

SPC is used to control the printer(s) in the spooler.

CSPFF is used to create format control files for a printer and is usually run by SYS or the person(s) responsible for administration of printers.

15.4 SPOOL: INSERT A FILE IN A QUEUE

When a file is inserted into a queue, the user can choose one or more options to enter after the file name, e.g.:

```
>SPOOL FILE NAME/option <CR>
```

If no extension is specified for the file name, the system will add the extension .PRN.

The following options are available:

1) F:n
This specifies a format code, where "n" is any number from 0 to 255. If no option is specified, the user's format code as specified by CONFIGU (see Section 6.4) is assigned.

2) P:n
This specifies the user priority, where n is any number from 0 to 255. If no option is specified, the user priority as defined by SYS is assigned.

3) H+
Orders a header page to start the printout. The header includes the user ID, the name of the file, the time the file was inserted in the queue and the time of printout. If no option is specified, a header page will be printed if the format code is less than 100.

4) H-
This option specifies that a header page is not to be printed.

5) L
This option leaves the file on the disk after printing. If /L is not specified, the file will be erased after it is printed out, if the original was in the root volume on the first physical drive (1:). If it is in any other volume, a copy will be made in the root volume, and the original will be erased. The copy in the root volume is stored under the name SQxxxxxx.SYS, where xxxxxx is a unique spool queue number generated by BRIDOS.

This option is not operative if the file has been sent to the spooler through the LST: device (see Section 15.2).

The above options should be entered on a single line, separated by slashes, e.g.

```
F>SPOOL INVOICES/F:91/P:50/H+/L <CR>
```

15.5 SPOOL: DISPLAY FILES IN QUEUE

A user can obtain a listing of the files which he/she currently has in the spool queue by entering:

```
>SPOOL <CR>

qn    ff   pri   hh:mm:ss   x

1     25   50    14:52:50   2

Enter queue number to abort job (^C to exit, < to redisplay)
```

In the above display,

qn	= Queue number
ff	= Format code
pri	= User priority
hh:mm:ss	= Time the file was submitted to the spool queue
x	= Number of the printer (if printing has started). No number is listed if printing has not started.

15.6 SPOOL: REMOVE A FILE FROM THE QUEUE

To remove a file from the queue, run SPOOL to obtain a listing as in Section 15.3. Enter the queue number of the file to be removed, followed by <CR>. A new listing will be displayed.

To return to the system, enter ^C.

15.7 ABORT LST:

This command is used to remove data from the list output file after they have been sent through the LST: device, but before they have been inserted in the spooler queue. Enter:

```
F>ABORTLST <CR>
```

This will remove all data which are still in the list output file and were submitted since the last SPOOL LST:, ABORTLST or GETLST.

15.8 GET LST:

This command is used to remove data that have been sent to the list output file through the LST: device. The data will be stored in the file and volume named in the command. If an extension is not given for the file name, the extension .PRN is added by default. If no volume letter is specified, the logged volume will be used.

Thus to remove data from the list output file (submitted since the last SPOOL LST:, ABORTLST or GETLST) and store them in the file KEEP in the volume HELLO, which is attached to C:, enter:

```
C>GETLST KEEP <CR>
```

15.9 PRINTER CONTROL: SPC

The SPC program is used to control the printers that are fed by the spooler. SPC is normally run at the start of a working day by the person(s) responsible for administration of printers.

In general, each printer is activated and a format code is then assigned. Any number of spool printers can be activated for the same format code. Each printer will process a file from the format code queue as soon as it is ready for a new file. Check that the paper in the printer corresponds to the format code.

A printer can be either active or inactive. When the system is started, all spooler printers are in deactivated status. Nothing can be printed. To activate a printer, run the program by entering:

```
+F>SPC <CR>

Printer 1 is not active
Printer 2 is not active

^C   Return to system
<    Display status again
n    Choose a printer (n = 1,2...)
Z    Display queues

Choose function: 1 <CR>
```

The command Z <CR> generates a display of the contents of each queue to facilitate selection of appropriate printers. When a printer has been selected, the program offers a new set of options.

```
*Choose function: 1 <CR>

Printer 1 is not active
Printer 2 is not active

Enter command for printer 1

^C   Return to system
<    Display status again
n    Choose a printer (n = 1,2...)
A    Activate the printer
Z    Display queue lengths

Choose function: A <CR>
Enter format code: 125 <CR>
Adjust paper in printer and press RETURN <CR>

Printer 1 is active and printing
Activated by GHB at 09:47:52 for format 125
Printing file LST:PDC.PRN submitted by PDC at 09:11:35
Printing of this file started at 09:47:55, priority code = 10

Printer 2 is not active

Enter command for printer 1

^C   Return to system
<    Display status again
n    Choose a printer (n = 1,2...)
D    Deactivate the printer after printing this file
I    Interrupt the printer

Choose function:
```

In the above example, printer 1 started printing as soon as it was activated and a format code was assigned. Obviously, there must have been at least one file for format code 125 in the spooler. The file LST:PDC.PRN was first in the queue.

Note that a report on the status of all printers can be obtained by running SPC. For example, the following display could be generated:

```
+A>SPC <CR>

Printer 1 is not active

Printer 2 is active and printing
Activated by GHB at 09:30:15 for format 115
Printing file LST:LGB.PRN submitted by LGB at 14:25:00
Printing of this file started at 15:26:35, priority code 51

Printer 3 is not active

^C   Return to system
<    Display status again
n    Choose a printer (n = 1,2...)
Z    Display queue lengths

Choose function: 2 <CR>

Printer 1 is not active

Printer 2 is active and printing
Activated by GHB at 09:30:15 for format 115
Printing file LST:LGB.PRN submitted by LGB at 14:25:00
Printing of this file started at 15:26:35, priority code 51

Printer 3 is not active

Enter command for printer 2

^C   Return to system
<    Display status again
n    Choose a printer (n = 1,2...)
D    Deactivate the printer after printing this file
I    Interrupt the printer

Choose function:
```

GHB can interrupt the printout on printer 2 (e.g. in order to replace the ribbon) by entering I <CR>. The following information will be displayed:

```
Printer 1 is not active

Printer 2 has been interrupted while printing
Activated by GHB at 09:30:15 for format 115
Printing file LST:LGB.PRN submitted by LGB at 14:25:00
Printing of this file started at 15:26:35, priority code 51
The printer was interrupted by GHB at 15:30:10

Printer 3 is not active

Enter command for printer 2

^C   Return to system
<    Display status again
n    Choose a printer (n = 1,2...)
D    Deactivate the printer immediately leaving this file in queue
Q    Abort this file and print next file in queue
C    Continue printing this file
R    Restart printing of this file from the beginning

Choose function:
```

If the printer is deactivated with the command D <CR> (e.g. in order to correct a fault), the printout will be restarted from the beginning on the same printer as soon as it is reactivated, if this was the only printer serving the format queue to which the file was submitted. If another printer was serving the same queue, the file will be printed on that printer as soon as the current printout is finished.

If printout of a file is aborted, the file is removed from the queue.

15.10 CSPFF: IMPLEMENTING FORMAT CODES

This utility is used to defined a string of characters which is sent to a printer to define a format. The string is sent a)when the printer is activated b) when a printout is restarted from the beginning of a file after an interruption (see below). The string may contain various printer commands, such as form length, print density and vertical or horizontal TAB stops. Consult your hardware supplier about the printer commands for your printer.

To run the program enter:

```
F>CSPFF
*
```

Enter a format command line after the asterisk. The command line must start with a legal format code (0-255), followed by either ++, +-, --, -+, followed by a space, followed by the characters that are to be sent to the printer.

The command ++ = mask high bit/expand TABS
The command +- = mask high bit/do not expand TABS
The command -- = do not mask high bit/do not expand TABS
The command -+ = do not mask high bit/expand TABS

The high-bit is the 8th bit in the 7-bit ASCII-code byte.

A format command line for a daisy-wheel printer might look like this:

*50+-<CR>

To insert control characters in the command line, enter ^ followed by the character(s) controlling the printer, e.g. to insert the command ^L type the character ^ and then the character L. Do not hold down the CONTROL key when you enter a control character in the command line.

To end a command line, enter <CR>.

To return to the system, enter <CR> after the *-prompt.

Each time CSPFF is run, it creates a file called PRINTER.FMT on drive A:. This file can be keyed to a printer by renaming it PRINTERn.FMT, where n is the printer number. Consult your hardware dealer about the printer number, which corresponds to a printer port on the central disk server.

If the printer-spooler cannot find a PRINTERn.FMT file (or if the file does not contain a command line for the format being activated) no format command will be sent to the printer, high bits will be masked and TABS will be expanded.

NOTE: If printing has been interrupted, restarting with the R-command will re-transmit the format command to the printer. The format command usually includes a sequence meaning SET TOP OF FORM. This means that you must reset the paper to the top of form position before entering the R-command.

The command string is re-transmitted because many printers lose formatting information when they are turned off. You can thus interrupt the printer, turn it off and correct a fault, restart the printer and restart the printout without respooling the file.

See Appendix 1 for a list of typical paper formats.

16.1 CONVERTING CP/M FILES TO BRIDOS FILES

BRIDOS is based on emulator technology. It creates an environment that is virtually identical to a given operating system - such as CP/M - so that programs written for that environment can also function under BRIDOS.

HOWEVER, THE DISKETTE FORMATS FOR CP/M AND FOR BRIDOS ARE VERY DIFFERENT. DO NOT - REPEAT NOT - TRY TO:

A) RUN A CP/M DISKETTE UNDER BRIDOS
B) RUN A BRIDOS DISKETTE UNDER CP/M.

To copy files from a CP/M diskette to a BRIDOS diskette, use the utility program TBD.

To copy files from a BRIDOS diskette to a CP/M diskette, use the utility FBD.

The TBD and FBD programs are modified for specific microcomputers. They are delivered with the names TBDxxxx or FBD xxxx, where XXXX is designates the type of microcomputer. These programs cannot be used to transfer files that are not CP/M-compatible.

This TBD and FBD programs can be used to transfer files between floppy disks or between a Winchester and a floppy disk.

THE TBD AND FBD PROGRAMS RUN UNDER CP/M. Insert a CP/M diskette in drive A: and a BRIDOS diskette in another drive. Load TBD (or FBD) through CP/M. An asterisk prompt will be displayed on the screen. The command syntax is the same as for PIP (see Section 14.0), including wild cards (general indicators).

TBD assumes that the source files are on a CP/M diskette and that the destination is a BRIDOS diskette.

FBD assumes that the source files are on a BRIDOS diskette and that the destination is a CP/M DISKETTE.

Either or both diskettes can be changed before entering a command.

Return to CP/M by entering ^C.

17.0 UTILITY PROGRAMS

The BRIDOS system includes a full set of utilities, which are listed below in Sections 17.1-??. Some of these utilities are not delivered with the single-user Uni-system.

BRIDOS utilities can operate with "*" and "?". The "*" indicates a string of characters and the "?" indicates a single character.

TEST*.* refers to all files with names beginning with TEST, followed by any string and any extension.

T???.TXT refers to all files ending in .TXT with a 4-character first name starting with T.

. refers to all files.

*.DOC refers to all files with the extension .DOC.

The "/" is used to specify options. If a utility program can accept options, they are specified by "/" and the appropriate character, e.g.:

```
A>DIR /P
```

is used to obtain a directory of all files on the A-drive, including file parameters. NOTE THAT THE SPACE BETWEEN THE DIR COMMAND AND /P MUST BE ENTERED WITH THE SPACE BAR.

If a utility is used without specifying a drive, the logged drive is assumed. In the Omni- and Multi-systems, any letter that has been attached to a volume can be specified.

Any utility program (or any other program) can be aborted by stopping screen output with ^S and then pressing ^C. If SUBMIT or THREAD is active, it will also be aborted.

17.1 VOLINI

VOLINI is used to initialize a diskette or hard disk for BRIDOS operation. The program records the name of the medium and the date of initialization. An empty directory is also created. Enter:

A>VOLINI <CR>

The program then asks for the name of the drive with the disk to be initialized. Enter the drive letter, e.g. D: and then RETURN.

The program then asks for the name of the volume. Enter a maximum of 32 characters followed by RETURN.

The program then asks for today's date, which is the creation date. Enter the date as 6 numeric characters in the form YYMMDD. If your computer has a built-in clock and BRIDOS has been installed to use it, VOLINI will automatically display the date.

VOLINI now asks if the volume is to be bootable. Enter Y or N for yes or no. In a non-bootable volume, the disk space normally occupied by the system is available for data files. An attempt to boot from a non-bootable volume will generate the error message: "Not a bootable volume."

OPTIONS: N

If the "N" option is entered, e.g. VOLINI /N <CR>, the program will allocate space for the system if a bootable volume has been requested, but will not put a system on the diskette. The volume created will be bootable but will not in fact boot until a system is moved on to the diskette.

17.2 SYSGEN

SYSGEN is used to copy the cold-start loader (IPL) and the system track(s) from one diskette to another, without affecting the files on the diskette. Enter **SYSGEN**.

The program then asks for the source drive. Enter the appropriate letter for the drive to be read. SYSGEN then asks for the target drive. Enter the appropriate letter.

The system can only be moved on to a bootable volume (see Section 17.1, N-option).

17.3 CONFIG

CONFIG is used to set up the cold-start environment. For Omni- and Multi-systems, see CONFIGU and CONFIGT, Sections 6.1-6.4. A CONFIG for Uni-systems will be available in June, 1983.

17.4 DIR

DIR is used to display the contents of a volume directory. The amount of free space is also displayed. Enter DIR followed by a drive name, a file name and an option, if necessary, and then RETURN.

If no drive is specified the logged drive is assumed

If a file name (with or without ? and *) is not specified, all files in the directory are listed.

OPTIONS: A, P, C

Entering the A-option, e.g. DIR B:*.TXT/A will generate a display of file attributes after the file name. There are three attributes: P, D and X. P indicates that the file is write-protected. D indicates that the file is a directory. X indicates that a file aais marked for exclusive use (see Section ??) and can only appear if the file is a directory file.

Entering the P-option, e.g. DIR B:*.TXT/P will generate a display of file type and file parameters after the file name. File types and parameters are listed below:

File type	Parameters
0 (index-seq file)	Record length, key length, sequencer length
1 (Secondary key support)	Record length, primary key length, secondary key length
2 (Primary key generator)	Record length, key length
8 (Sequential file, fixed record length)	Record length
9 (Sequential file, variable record length)	None
13 (Binary-load file)	None
15 (Sequential file, no record structure)	None

Entering the C-option, e.g. DIR B:*.TXT/C generates a display of CP/M-compatible files only.

The above options can be combined in a single command line.

17.5 REN

REN i s used to change the name of a file. To change the name of DOC.TXT on drive C: to NEW.TXT, enter REN C:NEW.TXT=DOC.TXT <CR>. Note that the new name is entered first.

17.6 ERA

ERA is used to erase a file. The space previously occupied by the file will be treated as free space by BRIDOS and can thus be allocated for other files. Enter ERA followed by a drive name if required and the name of the file or files. The symbols * and/or ? can be used.

The names of the files that have been erased are displayed on the screen. A write-protected file cannot be erased until the write-protect attribute has been removed with ENABLE (see Section 17.7). A directory cannot be erased unless it is empty. The root directory for a drive can never be erased.

If * or ? are used in an ERA command line, BRIDOS will ask for confirmation before it erases each file. Answer Y if the file is to be erased, N if not.

OPTION: N

Entering the option /N will enable the symbols * and ? to be used without having to confirm erasure of each file.

17.7 PROTECT AND ENABLE

PROTECT is used to write-protect a file so that it cannot be changed or erased. Enter PROTECT DOC.TXT <CR>.

ENABLE is used to remove the write-protect attribute from a file. Enter ENABLE DOC.TXT <CR>.

If * or ? are used in a command line with PROTECT or ENABLE, , BRIDOS will ask for confirmation before it changes th write-protect attribute. Answer Y if the atribute is to be changed, N if not.

OPTION: N

Entering the option /N will enable the symbols * and ? to be used without having to confirm changing the write-protect attribute for each file.

17.8 DUMP

DUMP is used to display the contents of a file on the screen. Enter DUMP, followed by drive name and the file name. The symbols ? and * cannot be used with this program.

The file will be displayed in ASCII format. For each record in the file, a record number and the contents of the record are displayed.

The first 3 bytes in each record of a CP/M-compatible file indicate the CP/M record number.

OPTIONS: H, C

Entering the H-option will display the contents in hex as well as ASCII format.

Entering the C-option is possible only for CP/M-compatible files and will display the relative addresses in each file, starting with 100 hex, instead of the record number.

The C- and H-options can be entered in the same command line.

17.9 TYPE

TYPE is used to display the contents of a CP/M-compatible file on the screen. Enter type by the drive name and the file name. To abort this program, press the space bar. The symbols ? and * cannot be used with this program.

17.10 SUBMIT

SUBMIT is used to instruct BRIDOS to read console input from a file instead of from the console. Enter SUBMIT followed by the drive name, the file name and optional parameters, e.g.

```
C>SUBMIT B:DOC
```

The parameters for this utility are exactly the same as for the SUBMIT function in CP/M, including XSUB.

If a file name without an extension is specified, BRIDOS will assume that the file has the extension SUB.

The file DOC.SUB will be read by BRIDOS as console output until it comes to the end of the file. Programs that read from the console through the buffered console input function will also receive input from this file.

If the characters "$n" (n = a decimal digit from 1 to 9) occur in the file, BRIDOS will replace these two characters with the corresponding parameter in the command line, e.g. "$4" will be replaced by the 4th parameter after the file name, "$6" by the 6th parameter, etc.

A file which is read as console input can include a SUBMIT command with another file name, with the same syntax as the above command line. BRIDOS will execute the second file until it comes to the end of the file. BRIDOS then returns to the first submit file and continues from where it left off. Nested SUBMITS can be executed down to five levels. Parameters can be passed from one SUBMIT to another, as in the command line above.

17.11 THREAD

THREAD has the same functions as SUBMIT, with the following exceptions:

Only the command line(s) and not the buffered console input is read from the file.

The command line(s) are not displayed on the screen.

If a new THREAD or SUBMIT is found in the THREAD file, the new file will be initiated without returning to the old file, i.e. nesting is not possible.

17.12 SAVE

SAVE is used to store the contents of the TPA (Transient Program Area) on a diskette or disk as a CP/M-compatible COM file. Enter **SAVE**, followed by the number of 256-byte blocks that are to be saved, followed by the drive name and the file name, e.g. **SAVE 6 D:DOC.COM**.

The save starts from address 100 (hex).

The symbols ? and * cannot be used with this program.

17.13 TRACOB

TRACOB is used to convert a CP/M-compatible COM-file image to a BRIDOS binary-load file (BLF). It is also used to convert COBOL-80 overlays to BRIDOS sequential files. A COM-file image can be executed directly through the CCP emulator, but BLF files load faster and occupy less disk space. A COBOL-80 program with overlays can be used directly with BRIDOS, but CHAIN and SEGMENT functions execute much faster if the program is linked with CSBDxx (see COBOL Interface Manual) and COM-files and the overlays are converted to BRIDOS sequential files.

Enter TRACOB, followed by the target-drive name and the file name without an extension. Both TRACOB and a file with this name and the extension COM should be on drive A, which should be the logged drive. An equivalent BLF file will be written on the drive specified in the command line. This file will have the extension BLF.

If there are any COBOL-80 overlay files with the extension Vxx on drive A, BRIDOS will write sequential-file equivalents on the drive specified in the command line. These files will be given the extension Vxx.

The names of all the files transferred are displayed on the screen.

17.14 ATTACH and DETACH

ATTACH and DETACH are used to assign and to free physical drives from BRIDOS directories (see Sections 9.1-10.1). These utilities are not included in the Uni-system.

APPENDIX 1: TYPICAL PAPER FORMATS

Format code	Form width	Form length	Char /inch	Lines /inch	Lines /page	Columns /page
0	80.5	11	10	6	66	80
1	80.5	11	12	6	66	96
2	80.5	11	16.5	6	66	132
3	80.5	11	10	8	88	80
4	8.5	11	12	8	88	96
5	8.5	11	16.5	8	88	132
6	11	11	10	6	66	110
7	11	11	10	8	88	110
8	13.2	11	10	6	66	132
9	13.2	11	10	8	88	132
10	8.5	11.5	10	6	69	80
11	8.5	11.5	12	6	69	80
12	8.5	11.5	16.5	6	69	132
13	8.5	11.5	10	8	92	80
14	8.5	11.5	12	8	92	96
15	8.5	11.5	16.5	8	92	132
16	11.5	11.5	10	6	69	110
17	11.5	11.5	10	8	92	110
18	13.2	11.5	10	6	69	132
19	13.2	11.5	10	8	92	132
20	8.5	12	10	6	72	80
21	8.5	12	12	6	72	96
22	8.5	12	16.5	6	72	132
23	8.5	12	10	8	96	80
24	8.5	12	12	8	96	96
25	8.5	12	16.5	8	96	132
26	12	12	10	6	72	110
27	12	12	10	8	96	110
28	13.2	12	10	6	72	132
29	13.2	12	10	8	96	132
30	11	8	10	6	48	110
31	11	8	10	8	64	110
32	11.7	8	10	6	48	110
33	11.7	8	10	8	64	117
34	13.2	8	10	6	48	132
35	13.2	8	10	8	64	132
40	11	8.5	10	6	51	110
41	11	8.5	10	8	68	110
42	11.7	8.5	10	6	51	117
43	11.7	8.5	10	8	68	117
44	13.2	8.5	10	6	51	132
45	13.2	8.5	10	8	68	132

TABLE OF CONTENTS

1. GENERAL INFORMATION

1.1 INTRODUCTION

This manual describes how the BRISAM file management system is called from an Z-80* assembly language program. An 8080 program can also be used but a Z-80 CPU must be used to execute the program. Each operation described in the BRISAM users manual is treated separately. The user is referred to that manual for a detailed description of each operation and for general considerations.

1.2 LINKAGE

BRISAM is called through the CP/M emulator by setting the C register to 255, setting a pointer to the parameter block in the DE register pair and executing a CALL 5 operation.

1.3 NOTATION

Z-80 assembler notation is used. Examples of parameter blocks and other sending and receiving fields are given.

1.4 STATUS

The status for each operation is returned in one byte pointed to by the second and third bytes of the parameter block. The status value is in binary representation.

1.5 VOLUME NUMBERS

If a logical volume number or disk drive number is called for, use 1 for A:, 2 for B: etc. In systems that don't support logical volume management this will refer to a physical drive number, in systems that support logical volumes this will refer to a logical volume number. When a physical drive number is specifically called for the physical drive number must be used.

* Z-80 is a registered trademark of ZILOG

2. PARAMETER BLOCK SYNTAX

2.1 GET FATAL ERROR CODE

```
GECPB:    DEFB      26          ; operation code
          DEFW      STATUS      ; pointer to where the serious error code
                                ; is to be stored
```

2.2 MOUNT VOLUME

```
MVPB:     DEFB      0           ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFB      DRIVE       ; disk drive or logical volume number
          DEFW      VID         ; pointer to where volume ID is to be
                                ; stored
;
;         the volume ID has the following format
;
VID:      DEFS      32          ; volume name in ASCII
          DEFS      6           ; creation date in ASCII (YYMMDD)
          DEFS      6           ; last backup date in ASCII (YYMMDD)
```

2.3 MOUNT VOLUME QUIETLY

```
MQPB:     DEFB      43          ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFB      DRIVE       ; disk drive or logical volume number
```

2.4 GET VOLUME IDENTITY

```
GVIDPB:   DEFB      27          ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFB      DRIVE       ; disk drive or logical volume number
          DEFW      VID         ; pointer to where volume ID is to be
                                ; stored (see MOUNT VOLUME)
```

2.5 DISMOUNT VOLUME

```
DVPB:     DEFB      1           ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFB      DRIVE       ; disk drive or logical volume number
                                ; 0 dismounts all drives
```

2.6 TEST MOUNT

```
TMPB:     DEFB      37          ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFB      DRIVE       ; disk drive or logical volume number
```

2.7 INITIALIZE DISK SYSTEM

```
IDSPB:    DEFB      42          ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFB      DRIVE       ; disk drive or logical volume number
                                ; 0 initializes all drives
```


2.8 CREATE FILE

```
CFPB:     DEFB     2          ; operation code
          DEFW     STATUS     ; pointer to where status is to be stored
          DEFB     DRIVE      ; disk drive or logical volume number
          DEFW     FILNAM     ; pointer to file name
          DEFB     RECLEN     ; record length (4 - 252)
          DEFB     KEYLEN     ; key length (see below)
          DEFB     FILTYP     ; file type
;
;         the file name is stored as follows
;
FILNAM:   DEFS     11         ; ASCII -- the extension is stored in the
                              ; last 3 positions, Unused positions are
                              ; filled by ASCII spaces (32)
;
;         the file types are as follows
;
;          0       index sequential file without secondary key support
;          1       index sequential file with secondary key support
;          2       index sequential file with primary key generation
;          8       sequential file with fixed record length
;          9       sequential file with variable record length
;         13       binary load file
;         15       sequential file without record organization
;
;          the key length byte has the following format
;
;     file type     key length byte disposition in bits
;                   bit 0 = least significant bit
;
;         0           5-0 key length, 7-6 sequencer length
;         1           1-0 primary key length, 7-2 secondary key length
;         2           1-0 key length
;       others        not used
;
```

2.9 ERASE FILE

```
EFPB:     DEFB     3          ; operation code
          DEFW     STATUS     ; pointer to where status is to be store
          DEFB     DRIVE      ; disk drive or logical volume number
          DEFW     FILNAM     ; pointer to file name
;
;         the file name is stored as follows
;
FILNAM:   DEFS     11         ; ASCII -- the extension is stored in th
                              ; last 3 positions, Unused positions are
                              ; filled by ASCII spaces (32)
```


2.10 OPEN FILE

```
OFPB:     DEFB      5          ; operation code
          DEFW      STATUS     ; pointer to where status is to be stored
          DEFB      DRIVE      ; disk drive or logical volume number
          DEFW      FILNAM     ; pointer to file name
          DEFW      LFNO       ; pointer to where the resulting one byte
                               ; logical file number is to be stored
;
;         the file name is stored as follows
;
FILNAM:   DEFS      11         ; ASCII -- the extension is stored in the
                               ; last 3 positions, Unused positions are
                               ; filled by ASCII spaces (32)
```

2.11 OPEN FILE FOR READ ONLY

```
OROPB:    DEFB      46         ; operation code
          DEFW      STATUS     ; pointer to where status is to be stored
          DEFB      DRIVE      ; disk drive or logical volume number
          DEFW      FILNAM     ; pointer to file name
          DEFW      LFNO       ; pointer to where the resulting one byte
                               ; logical file number is to be stored
;
;         the file name is stored as follows
;
FILNAM:   DEFS      11         ; ASCII -- the extension is stored in the
                               ; last 3 positions, Unused positions are
                               ; filled by ASCII spaces (32)
```

2.12 OPEN DIRECTORY

```
ODPB:     DEFB      47         ; operation code
          DEFW      STATUS     ; pointer to where status is to be stored
          DEFB      DRIVE      ; disk drive or logical volume number
          DEFW      LFNO       ; pointer to where the resulting one byte
                               ; logical file number is to be stored
```

2.13 OPEN FILE 32

```
O32PB:    DEFB      70         ; operation code
          DEFW      STATUS     ; pointer to where status is to be stored
          DEFB      DRIVE      ; disk drive or logical volume number
          DEFW      FILNAM     ; pointer to file name
          DEFW      LFNO       ; pointer to where the resulting one byte
                               ; logical file number is to be stored
;
;         the file name is stored as follows
;
FILNAM:   DEFS      11         ; ASCII -- the extension is stored in the
                               ; last 3 positions, Unused positions are
                               ; filled by ASCII spaces (32)
```


2.14 OPEN FILE WITH WRITE ENABLE

```
OWEPB:    DEFB      71          ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFB      DRIVE       ; disk drive or logical volume number
          DEFW      FILNAM      ; pointer to file name
          DEFW      LFNO        ; pointer to where the resulting one byte
                                ; logical file number is to be stored
;
;         the file name is stored as follows
;
FILNAM:   DEFS      11          ; ASCII -- the extension is stored in the
                                ; last 3 positions, Unused positions are
                                ; filled by ASCII spaces (32)
```

2.15 GET FILE PARAMETERS

```
GFIDPB:   DEFB      28          ; operation code
          DEFW      STATUS      ; pointer to where status is to be store
          DEFB      LFNO        ; the logical file number
          DEFW      FID         ; pointer to where file id is to be stored
;
;         the file identity format
;
FID:
UNO:      DEFS      1           ; physical drive number
RECLEN:   DEFS      1           ; record length
KEYLEN:   DEFS      1           ; key length (see CREATE FILE)
TYPE:     DEFS      1           ; file type and attributes
                                ; see CREATE FILE
```

2.16 CLOSE FILE

```
CFPB:     DEFB      6           ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFB      LFNO        ; the logical file number of the file to
                                ; be closed (0 closes all files)
```

2.17 EMPTY FILE

```
EFPB:     DEFB      4           ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFB      LFNO        ; the logical file number of the file to
                                ; be emptied
```

2.18 RENAME FILE

```
RENPB:    DEFB      40          ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFB      DRIVE       ; disk drive or logical volume number
          DEFW      OFNPTR      ; pointer to old file name
          DEFW      NFNPTR      ; pointer to new file name
```

2.19 SET FILE ATTRIBUTES

```
SFAPB:    DEFB      44          ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFB      DRIVE       ; disk drive or logical volume number
          DEFW      FNPTR       ; pointer to file name
          DEFB      NKL         ; new key length
          DEFB      NFT         ; new file type
```


2.20 MOVE FILE

```
MFPB:   DEFB    69          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    OLDDRV      ; old disk drive or logical volume number
        DEFB    NEWDRV      ; new disk drive or logical volume number
        DEFW    OLDFNP      ; pointer to old file name
        DEFW    NEWFNP      ; pointer to new file name
;
;       File names follow the same conventions specified above
;
```

2.21 GET FILE SIZE

```
GFSZPB: DEFB    63          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number of the file
        DEFW    FSZPTR      ; pointer to where the file size is to be
                            ; stored
```

2.22 INSERT RECORD

```
IRPB:   DEFB    16          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    RBPTR       ; a pointer to the record buffer
```

2.23 INSERT RECORD WITH SEQUENCING

```
ISPB:   DEFB    29          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    RBPTR       ; a pointer to the record buffer
```

2.24 DELETE RECORD

```
DRPB:   DEFB    17          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    RBPTR       ; a pointer to the record buffer
```

2.25 UPDATE RECORD

```
URPB:   DEFB    23          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    RBPTR       ; a pointer to the record buffer
```

2.26 GET EQU

```
GEPB:   DEFB    18          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    RBPTR       ; a pointer to the record buffer
```

2.27 GET GRE EQU

```
GGEPB:  DEFB    19          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    RBPTR       ; a pointer to the record buffer
```


2.28 GET GRE

```
GGPB:   DEFB    20          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    RBPTR       ; a pointer to the record buffer
```

2.29 GET LESS EQU

```
GLEPB:  DEFB    21          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    RBPTR       ; a pointer to the record buffer
```

2.30 GET LESS

```
GLPB:   DEFB    22          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    RBPTR       ; a pointer to the record buffer
```

2.31 GET GRE WITH LOOK AHEAD READING

```
GGLAPB: DEFB    45          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    RBPTR       ; a pointer to the record buffer
```

2.32 GET USING SECONDARY KEY

```
GSKPB:  DEFB    38          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    RBPTR       ; a pointer to the record buffer
```

2.33 READ SEQUENTIAL RECORD

```
RSRPB:  DEFB    13          ; operation code
        DEFW    STATUS      ; pointer to where status is to be store
        DEFB    LFNO        ; the logical file number
        DEFW    RBPTR       ; a pointer to the record buffer
        DEFW    RLPTR       ; a pointer to where the one byte record
                            ; length is to be stored
```

2.34 WRITE SEQUENTIAL RECORD

```
WSRPB:  DEFB    14          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    RBPTR       ; a pointer to the record buffer
        DEFB    RECLEN      ; record length (not used with fixed
                            ; record length files)
```

2.35 SET BEGINNING OF FILE

```
BOFPB:  DEFB    11          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
```


2.36 SET END OF FILE

```
EOFPB:  DEFB    12          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
```

2.37 GET POSITION

```
GPPB:   DEFB    9           ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    POSVEC      ; a pointer to where a four byte position
                            ; vector that identifies the current file
                            ; position is to be stored
```

2.38 SET POSITION

```
SPPB:   DEFB    10          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    POSVEC      ; a pointer to a four byte position
                            ; vector previously returned by
                            ; "GET POSITION"
```

2.39 READ SEQUENTIAL

```
RSPB:   DEFB    7           ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    RBPTR       ; a pointer to the record buffer
        DEFB    XFRLEN      ; the number of bytes to be read
        DEFW    LXFRDP      ; a pointer to where the number of bytes
                            ; actually read is to be stored
```

2.40 WRITE SEQUENTIAL

```
WSPB:   DEFB    8           ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    RBPTR       ; a pointer to the record buffer
        DEFB    XFRLEN      ; the number of bytes to be written
        DEFW    LXFRDP      ; a pointer to where the number of bytes
                            ; actually written is to be stored
```

2.41 LOAD FILE TO MEMORY

```
LDPB:   DEFB    15          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    EPTAB       ; a pointer to where the entry point
                            ; table is to be stored
;
;       the entry point table has the following format
;
EPTAB:  DEFB    MAXEP       ; max number of entry points to be read
        DEFS    1           ; the number of entry points in the file
        DEFS    2*MAXEP     ; the entry point addresses read from
                            ; the file
```

2.42 LOAD AND RUN

```
LRPB:   DEFB    34      ; operation code
        DEFW    STATUS  ; pointer to where status is to be stored
        DEFB    DRIVE   ; disk drive or logical volume number
        DEFW    FNAME   ; pointer to file name
        DEFW    STAPTR  ; stack pointer value
```

2.43 START TRANSACTION

```
STPB:   DEFB    24      ; operation code
        DEFW    STATUS  ; pointer to where status is to be stored
```

2.44 END TRANSACTION

```
ETPB:   DEFB    25      ; operation code
        DEFW    STATUS  ; pointer to where status is to be stored
```

2.45 GET TRANSACTION STATUS

```
GTSPB:  DEFB    41      ; operation code
        DEFW    STATUS  ; pointer to where status is to be stored
```

2.46 READ DISK PAGE

```
RDPPB:  DEFB    30      ; operation code
        DEFW    STATUS  ; pointer to where status is to be stored
        DEFB    DRIVE   ; physical disk drive number
        DEFW    PAGENO  ; physical page number
        DEFW    BUFFER  ; a pointer to where the data is stored
```

2.47 WRITE DISK PAGE

```
WDPPB:  DEFB    31      ; operation code
        DEFW    STATUS  ; pointer to where status is to be stored
        DEFB    DRIVE   ; physical disk drive number
        DEFW    PAGENO  ; physical page number
        DEFW    BUFFER  ; a pointer to where the data is read
```

2.48 GET NUMBER OF PAGES ON DRIVE

```
GNOPPB: DEFB    32      ; operation code
        DEFW    STATUS  ; pointer to where status is to be stored
        DEFB    DRIVE   ; physical disk drive number
        DEFW    PNOPTR  ; pointer to where number of pages is st re
```

2.49 GET PAGE SIZE

```
GPSPB:  DEFB    33      ; operation code
        DEFW    STATUS  ; pointer to where status is to be stored
        DEFB    DRIVE   ; physical disk drive number
        DEFW    PSPTR   ; pointer to where page size is stored
```

2.50 GET FREE SPACE

```
GFSPB:  DEFB    39      ; operation code
        DEFW    STATUS  ; pointer to where status is to be stored
        DEFB    DRIVE   ; physical disk drive number
        DEFW    FSPTR   ; pointer to where free space is stored
```


2.51 GET DRIVE ATTRIBUTES

```
GDRAPB:   DEFB      53          ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFB      DRIVE       ; physical disk drive number
          DEFW      FSPTR       ; pointer to where drive attributes are
                                ; stored
;
;         the drive attributes are stored in one byte
;         the bits have the following meaning
;
;         bit 0     media test to be performed when initializing volume
;         bit 1     non removable media on drive
;         bit 2     big directories (> 255 files)
;         bit 3     drive supports bootable volumes
;
```

2.52 GET SYSTEM TRACK PARAMETERS

```
GSTPAR:   DEFB      54          ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFB      DRIVE       ; physical disk drive number
          DEFW      STPPTR      ; pointer to where the system track params
                                ; are to be stored
;
;         the system track parameters are stored in 5 bytes as follows
;
;         DEFS      1           ; the number of pages on the system track(s)
;         DEFS      2           ; the first system track page number
;         DEFS      2           ; the length of the boot sector in bytes
;
```

2.53 READ BOOT SECTOR

```
RBSPB:    DEFB      55          ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFB      DRIVE       ; physical disk drive number
          DEFW      BUFPTR      ; pointer to where to boot sector contents
                                ; are to be stored
```

2.54 WRITE BOOT SECTOR

```
WBSPB:    DEFB      56          ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFB      DRIVE       ; physical disk drive number
          DEFW      BUFPTR      ; pointer to a buffer containing the data
                                ; to be written to the boot sector
```

2.55 SHUT DOWN SYSTEM

```
SDSPB:    DEFB      68          ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
```

2.56 GET DATE

```
GDPB:     DEFB      35          ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFW      DATPTR      ; pointer to where date is to be stored
```


2.57 SET DATE

```
SDPB:   DEFB    36          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFW    DATPTR      ; pointer to date
```

2.58 GET TIME

```
GTPB:   DEFB    57          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFW    TIMPTR      ; pointer to where time is to be stored
```

2.59 SET TIME

```
STPB:   DEFB    58          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFW    TIMPTR      ; pointer to time
```

2.60 ATTACH

```
ATTPB:  DEFB    48          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    FVN         ; father volume number
                            ; bit 7 = 1 if physical drive
        DEFB    NVN         ; new volume number
                            ; bit 7 = 1 if exclusive use
                            ; bit 6 = 1 if read only
        DEFW    DNPTR       ; pointer to directory name
```

2.61 DETACH

```
DETPB:  DEFB    49          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LVNO        ; logical volume number
```

2.62 GET LOGICAL VOLUME IDENTITY

```
GLVIPB: DEFB    52          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LVNO        ; logical volume number
        DEFW    LVIPTR      ; pointer to where the logical volume
                            ; identity is to be stored
;
;       the logical volume identity is stored in 13 bytes as follows
;
;       DEFS    1           ; logical volume number of father volume
;                           ; if bit 7 = 1 then bits 0 to 4 are the
;                           ; physical drive number and this is a root
;                           ; volume
;       DEFS    11          ; directory name if not root volume
;       DEFS    1           ; bit 0 = 1 if read only
;
```


2.63 GET RUN UNIT

```
GRUPB:  DEFB    50          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFW    RUPTR       ; pointer to where run unit code is stored
;
;       the run unit code is stored as 2 ASCII numeric
;       characters "01" to "99"
;
```

2.64 RESET RUN UNIT

```
RRUPB:  DEFB    51          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
```

2.65 LOCK FILE

```
LFPB:   DEFB    59          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
```

2.66 LOCK RECORD

```
LRPB:   DEFB    60          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    KEYPTR      ; pointer to the key
```

2.67 UNLOCK FILE

```
UFPB:   DEFB    61          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
```

2.68 UNLOCK RECORD

```
URPB:   DEFB    62          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    LFNO        ; the logical file number
        DEFW    KEYPTR      ; pointer to the key
```

2.69 STORE MESSAGE

```
SMPB:   DEFB    64          ; operation code
        DEFW    STATUS      ; pointer to where status is to be stored
        DEFB    HOSTAD      ; host address (0 for this version)
        DEFW    MBADDR      ; message buffer address
;
;           the message buffer has the following format
;
MBADDR: DEFB    MBEND-$-1   ; message length not inc length byte
        DEFB    MTYPE       ; message type (16 - 31)
        DEFB    SENDER      ; run unit of sender
        DEFS    3           ; user id of RECEIVER
        DEFS    ML          ; message body
MBEND   EQU     $
```



```
;
;         The key is MTYPE, SENDER and RECEIVER
;         SENDER is automatically filled in by the system
;         Reserved message types are:
;
;              16 -- logon message where SENDER = run unit and
;                    RECEIVER is user id of logged on user
;
;              17 -- spooler status message where RECEIVER = SP1
;                    for spooler 1, SP2 for spooler 2 etc.
;
;              18 -- message to spooler used by SPC
;
;              19 -- 23 reserved for future system utilities
;
```

2.70 GET MESSAGE

```
GMPB:     DEFB      65          ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFB      HOSTAD      ; host address (0 for this version)
          DEFW      MBADDR      ; message buffer address
;
;         For GET MESSAGE a value of 255 in a key byte will match
;         anything (wild card). The message length is filled in
;         by the system when a message is found.
;
```

2.71 UPDATE MESSAGE

```
GMPB:     DEFB      66          ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFB      HOSTAD      ; host address (0 for this version)
          DEFW      MBADDR      ; message buffer address
```

2.72 ERASE MESSAGE

```
EMPB:     DEFB      67          ; operation code
          DEFW      STATUS      ; pointer to where status is to be stored
          DEFB      HOSTAD      ; host address (0 for this version)
          DEFW      MBADDR      ; message buffer address
;
;         only the key is used with ERASE MESSAGE
;
```


APPENDIX A

INDEX OF OPERATIONS IN OP CODE ORDER

BRISAM COBOL-80
Interface Manual

November 8, 1982

Gerald Berger
Stockholm, Sweden

INTRODUCTION

This manual describes how the BRISAM file management system is used by a COBOL-80* program. There are two ways to use BRISAM with COBOL-80, through the "CALL USING" interface where the BRISAM file management primitives can be invoked directly, and through the "Library Replacement Package" where the COBOL-80 Library routines which execute file management are replaced by routines which use BRISAM files. The "CALL USING" interface gives smaller programs but the "Library Replacement Package" gives standard COBOL-80 programs. Either way you will experience an enormous improvement in throughput compared to COBOL-80's standard library in combination with CP/M*. The BRISAM COBOL-80 Benchmarks give an example of how your COBOL-80 programs can perform if you use BRISAM. It is possible to use both methods in the same program.

A library of routines to access BRIDOS functions from a COBOL-80 program is also included. This is stored in the file BDLIB.REL and described in the last section of this manual.

THE "CALL USING" INTERFACE

This interface enables the COBOL-80 programmer to call the BRISAM primitives from a COBOL-80 program. Each operation described in the BRISAM users manual is treated separately. The user is referred to that manual for a detailed description of each operation and for general considerations.

LINKAGE

The interface routines are contained in the file COBINT.REL which is found on your distribution diskette. This file should be included when linking your object code modules and COBLIB with L-80.

NOTATION

All COBOL reserved words are shown in boldface and all user supplied entries are shown in normal type. The invocation syntax is shown first followed by an example of the data names used. Make sure that the usage (COMP,DISPLAY) matches that shown in the examples.

STATUS

A special operation is included to define which COBOL variable will receive the operation status on operation completion.

```
CALL "FHSETP" USING STATUS.
77 STATUS PICTURE 99 USAGE COMP.
```

After a call as indicated above the variable STATUS will receive the BRISAM status value on the completion of each operation until a new call to "FHSETP" redefines the status variable.

SERIOUS ERROR TRAP

A special operation may be used to trap on serious error conditions (those that return STATUS = 1). This operation is called with 2 parameters that define the screen position where the error message is to be displayed. Once enabled the trap will be in effect until the program is loaded again or a new program is loaded with $CHAIN. After the error message has been displayed the running program is aborted after the operator presses any key on the keyboard.

```
CALL "FHTRAP" USING LINE COLUMN.
77 LINE PICTURE 99 USAGE COMP VALUE 24.
77 COLUMN PICTURE 99 USAGE COMP VALUE 1.
```


SEGMENTATION AND CHAINING

The utility program TRACOB can be used to transfer a COBOL COM file and all its overlays (Vxx files) from CP/M format on drive A to BRISAM sequential file format on another drive by following the following procedure:

1. Insert the diskette with TRACOB.COM and the COBOL program(s) to be transferred in drive A.

2. Insert the BRISAM volume which is to receive the files on another drive.

3. Type: "TRACOB x:filename<"
   Where x is the drive (B, C etc.) containing the BRISAM volume which is to receive the BRISAM format files, filename is the name of the program and < is a carriage return.

Under BRIDOS the segmentation and chain facility will work using CP/M compatible files and the standard COBOL run time library. BRISAM sequential files can be used to cut program load time and save disk space. The module CSBD40.REL (ver 4.0) or CSBD46.REL (ver 4.6) must be linked to the program invoking the chain or overlays (before COBLIB.REL) for segmentation and chaining to work with BRISAM sequential files. (It replaces the module CSCPM in COBLIB)

"MOUNT VOLUME"

```
CALL "FHMOUN" USING  DISK-DRIVE  VOLUME-ID.

77 DISK-DRIVE PICTURE  X.        (disk drive = "A", "B", etc.)
01 VOLUME-ID.
        05  VOLUME-NAME  PICTURE  X(32).
        05  CREATION-DATE  PICTURE  X(6).
        05  LAST-BACKUP-DATE  PICTURE  X(6).
```

"DISMOUNT VOLUME"

```
CALL "FHDISM" USING DISK-DRIVE.

 77 DISK-DRIVE PICTURE  X.        (disk drive="A", "B", etc.)
```

"CREATE FILE"

```
CALL "FHCRTE" USING DISK-DRIVE FILE-NAME REC-LEN KEY-LEN TYPE.

77 DISK-DRIVE PICTURE X.         (disk drive = "A", "B", etc.)
77 FILE-NAME PICTURE X(11).
77 REC-LEN PICTURE 999 USAGE COMP.
77 KEY-LEN PICTURE 999 USAGE COMP.
77 TYPE PICTURE 999 USAGE COMP.
*
*    KEY-LEN specifies key and sequencer lengths as follows:
*    file type 0 -- sequencer length * 64 + key length
*    file type 1 -- secondary key length * 4 + primary key length
*    file type 2 -- primary key length
*
```

"ERASE FILE"

```
CALL "FHERAS" USING DISK-DRIVE FILE-NAME.

77 DISK-DRIVE PICTURE X.         (disk drive = "A", "B", etc.)
77 FILE-NAME PICTURE X(11).
```

"OPEN FILE"

```
CALL "FHOPEN" USING DISK-DRIVE FILE-NAME LOG-FNO REC-BUF.

77 DISK-DRIVE PICTURE X.         (disk drive = "A", "B", etc.)
77 FILE-NAME PICTURE X(11).
77 LOG-FNO PICTURE 99 USAGE COMP.
01 REC-BUF.
*
*        a description of a data record for the file follows here
*
```

Note that the "OPEN FILE" call assigns a data buffer to a file in the normal COBOL manner. This buffer is then automatically used by the record oriented operations when the file is referenced.

"CLOSE FILE"

```
CALL "FHCLOS" USING LOG-FNO.

77 LOG-FNO PICTURE 99 USAGE COMP.
```


"EMPTY FILE"

```
CALL "FHEMPT" USING LOG-FNO.
77 LOG-FNO PICTURE 99 USAGE COMP.
```

"INSERT RECORD"

```
CALL "FHINS" USING LOG-FNO.
77 LOG-FNO PICTURE 99 USAGE COMP.
```

"INSERT RECORD WITH SEQUENCING"

```
CALL "FHINSQ" USING LOG-FNO.
77 LOG-FNO PICTURE 99 USAGE COMP.
```

"DELETE RECORD"

```
CALL "FHDEL" USING LOG-FNO.
77 LOG-FNO PICTURE 99 USAGE COMP.
```

"UPDATE RECORD"

```
CALL "FHUPD" USING LOG-FNO.
77 LOG-FNO PICTURE 99 USAGE COMP.
```

"GET EQU"

```
CALL "FHGEQ" USING LOG-FNO.
77 LOG-FNO PICTURE 99 USAGE COMP.
```

"GET GRE EQU"

```
CALL "FHGGE" USING LOG-FNO.
77 LOG-FNO PICTURE 99 USAGE COMP.
```

"GET GRE"

```
CALL "FHGGR" USING LOG-FNO.
77 LOG-FNO PICTURE 99 USAGE COMP.
```

"GET LESS EQU"

```
CALL "FHGLE" USING LOG-FNO.
77 LOG-FNO PICTURE 99 USAGE COMP.
```

"GET LESS"

```
CALL "FHGLS" USING LOG-FNO.
77 LOG-FNO PICTURE 99 USAGE COMP.
```


"GET USING SECONDARY KEY"

```
CALL "FHGUSK" USING LOG-FNO.

77 LOG-FNO PICTURE 99 USAGE COMP.
```

"READ SEQUENTIAL RECORD"

```
CALL "FHRSQR" USING LOG-FNO REC-LEN.

77 LOG-FNO PICTURE 99 USAGE COMP.
77 REC-LEN PICTURE 999 USAGE COMP.
```

"WRITE SEQUENTIAL RECORD"

```
CALL "FHWSQR" USING LOG-FNO REC-LEN.

77 LOG-FNO PICTURE 99 USAGE COMP.
77 REC-LEN PICTURE 999 USAGE COMP.
```

"SET BEGINNING OF FILE"

```
CALL "FHSBOF" USING LOG-FNO.

77 LOG-FNO PICTURE 99 USAGE COMP.
```

"SET END OF FILE"

```
CALL "FHSEOF" USING LOG-FNO.

77 LOG-FNO PICTURE 99 USAGE COMP.
```

"GET POSITION"

```
CALL "FHGPOS" USING LOG-FNO POS-VEC.

77 LOG-FNO PICTURE 99 USAGE COMP.
77 POS-VEC PICTURE X(4).
```

"SET POSITION"

```
CALL "FHSPOS" USING LOG-FNO POS-VEC.

77 LOG-FNO PICTURE 99 USAGE COMP.
77 POS-VEC PICTURE X(4).
```

"START TRANSACTION"

```
CALL "FHBEG".
```

"END TRANSACTION"

```
CALL "FHEND".
```

"GET ERROR CODE"

```
CALL "FHGERR".
```

Note that this operation returns the fatal error code from the last operation that produced a fatal error (status = 1) into the status variable defined at the last call to "FHSETP".

THE LIBRARY REPLACEMENT PACKAGE

INTRODUCTION

This section describes how the COBOL-80 library replacement is to be used. The library replacement consists of several REL files that replace corresponding modules in MICROSOFT's standard release of the COBOL-80 run time library. There are a number of things that should be taken into consideration when changing over from the standard library to the BRISAM library.

- The maximum record length for this release is as follows:
  Index files -- no limit
  Relative files -- 250 bytes
  Sequential files -- no limit
  Line sequential files -- no limit

- Index, Relative, and Sequential files do not use CP/M compatible file formats. This should not make any difference since these file types were not accessible from any other language than COBOL-80 with the standard release of the library. Line sequential files do use CP/M compatible file formats and thus can be used for information transfer with other CP/M products. Their function is slightly modified to facilitate such transfer (see discussion of Line sequential files below). The index files produced by the BRISAM library replacement are interchangeable with those produced by the BRISAM COBOL-80 "CALL USING" interface and with those produced by the BRISAM MICROSOFT BASIC COMPILER interface. See discussion below concerning file interchangeability.

LINKAGE

The library replacement (REL files) routines included are:

- BDIO40 (ver 4.0), BDIO46 (ver 4.6) which contain the actual file management routines.
- BACPDAT (replaces ACPDAT) which contains the routines for ACCEPT FROM DATE, DAY and TIME. This module will work with versions 4.0 and 4.6.
- CSBD40 (ver 4.0), CSBD46 (ver 4.6)) which contain the routines for CHAIN and SEGMENTATION (overlay) management.

To use the BRISAM routines instead of the standard CP/M routines they must be loaded by L80 before the standard library. The routines being replaced will then not be loaded since the undefined names have already been defined by the BRISAM interface routines. Use BDIOxx (where xx = 40 or 46 depending on COBOL-80 version) if only the file management is to be replaced. Use BACPDAT if the date and time functions are to be replaced. Use CSBDxx (where xx is ver no) if the CHAIN and SEGMENTATION facilities should load from BRISAM BLF files and converted overlay sequential files (see discussion below).

An example link of the program "TEST" using all BRISAM alternatives and COBOL-80 version 4.6 follows below:

```
L80 TEST/N,TEST,BDIO46,BACPDAT,CSBD46/E
```


You can of course use LIB80 to make a new library so you don't have to specify so many files when linking.

SEGMENTATION AND CHAINING
The utility program TRACOB can be used to transfer a COBOL COM file and all its overlays (Vxx files) from CP/M format on drive A to BRISAM sequential file format on another drive by following the procedure below:

1. Insert the diskette with TRACOB.COM and the COBOL program(s) to be transferred in drive A.

2. Insert the BRISAM volume which is to receive the files on another drive.

3. Type: "TRACOB x:filename<"
   Where x is the drive (B, C etc.) containing the BRISAM volume which is to receive the BRISAM format files, filename is the name of the program and < is a carriage return.

Under BRIDOS the segmentation and chain facility will work using CP/M compatible files and the standard COBOL run time library. BRISAM sequential files can be used to cut program load time and save disk space. The module CSBDxx.REL must be linked to the program invoking the chain or overlays (before COBLIB.REL) for segmentation and chaining to work with BRISAM sequential files. (It replaces the module CSCPM in COBLIB)

FILE FORMATS
Index files are stored in standard BRISAM index files. The key item or group item will always be stored first in each record of the file even if it is declared in another position in the COBOL program. This transposition will be transparent to the COBOL programmer but should be remembered if the file is to be accessed from another language. If the record length is greater than 252 then each COBOL record will be stored in 2 or more BRISAM records depending on the record length. A sequencer byte will be appended to the COBOL key to give the BRISAM key.

Relative files are stored in BRISAM index files using the 2 byte relative record number as the BRISAM record key. This gives large savings in disk space utilization when using sparsely populated relative files.

Sequential files are stored in BRISAM sequential files. If the max record length is >= 252 bytes then the library replacement package will use file type 15 and maintain its own logical records keeping the same format as the standard COBOL-80 library, a two byte record length (msb first) followed by the record data.

Line sequential files are stored in CP/M compatible BRISAM index files (see BRIDOS manual). They are optimized for free format information transfer with other CP/M compatible program products. When a line sequential record is written all trailing spaces are removed. When a line sequential record is read the record buffer is padded with spaces after the physical end of record. Each record is ended with a carriage return line feed character combination and the last CP/M physical record in the file is padded with control Z characters. Line sequential files produced by the standard COBOL-80 library do not have to be converted before they can be read through the BRISAM library replacement

package.

FILE COMPATIBILITY WITH OTHER BRISAM LANGUAGE INTERFACES

The files produced through the BRISAM library replacement are totally compatible with those produced through the COBOL-80 "CALL USING" interface as long as the record length is < 252 bytes. Remember that the key is always stored in the first positions in each record in the file. The files are also compatible with those produced by the BASIC COMPILER INTERFACE if display format is used. COBOL stores variables with usage COMP as two binary bytes, the same as the BASIC compiler but COBOL stores the most significant byte first while BASIC stores the least significant byte first. If variables with usage COMP are to be accessed from BASIC then the byte order must first be reversed by an Assembler routine. See the MICROSOFT COBOL-80 documentation for a description of the storage format for COMP-3 variables.

TRANSACTION MANAGEMENT

The Library Replacement Package generates a "START TRANSACTION" operation whenever a WRITE, REWRITE or DELETE verb is executed if there isn't a transaction in progress already. The verbs OPEN, CLOSE and ACCEPT (from console) will generate "END TRANSACTION" operations if a transaction is in progress. Displaying more than 255 characters on the screen without executing a file access will also generate an "END TRANSACTION" operation. The OPEN INPUT verb will open the file for read only so that the file opened for input will not participate in transaction locks. Explicit "START TRANSACTION" operations can be generated by:

```
CALL "STRTRA"
```

Explicit "END TRANSACTION" operations can be generated by:

```
CALL "ENDTRA"
```


THE LIBRARY FOR BRIDOS SPECIAL FUNCTION ACCESS -- BDLIB

The library BDLIB contains routines for accessing BRIDOS special functions. Each routine in the library is described below.

GCUSER -- GET CURRENT USER

Invocation syntax:

```
CALL "GCUSER" USING USER.
77  USER PIC X(3) USAGE DISPLAY.
```

This routine will store the 3 character user id for the currently logged on user in the variable USER. If the system doesn't support access control then spaces will be stored.

SPOLST -- SUBMIT LST: FILE TO SPOOLER

Invocation syntax:

```
CALL "SPOLST" USING QUEUE PRIORITY.
77   QUEUE     PIC 9(5) USAGE COMP.
77   PRIORITY  PIC 9(5) USAGE COMP.
```

This routine will send the spool file built by sending characters to the list device configured for file output to the spooler for printing. The variable QUEUE should be set to the queue number to be used for printing this file and the variable PRIORITY should be set to the desired priority.

SETWSC -- SET WARM START COMMAND

Invocation syntax:

```
CALL "SETWSC" USING COMMAND-STRING.
77   COMMAND-STRING PIC X(32) USAGE DISPLAY.
```

This routine will set the specified command string into the system so that it will be executed on the next warm start. The command should use the same syntax as would be typed into the CCP. This routine can be used to run a BRIDOS utility from a COBOL program and to return to the COBOL program on completion of the utility. An example follows:

The COBOL program MAINMENU wants to run the BRIDOS utility SPC and when SPC is completed (when the operator presses ^C) to run MAINMENU again.

```
77 MAIN-MENU-COM PIC X(32) USAGE IS DISPLAY VALUE "MAINMENU".
    ..........
CALL "SETWSC" USING MAIN-MENU-COM.
CHAIN "A:SPC".
```

It should be noted that this function will not work if the program is invoked through a THREAD unless the program invocation is the last command in the THREAD file and is not followed by a CR. It will work if the program is invoked through SUBMIT but the utilities will read input from the SUBMIT file. Note also that the BRIDOS CHAIN module CSBDxx must be specified at link time if you want to CHAIN to a BLF file.

BRISAM BASCOM
Interface Manual

March 18, 1983

INTRODUCTION

This manual describes how the BRISAM file management system is called from a Microsoft BASCOM (BASIC compiler) program. Each operation described in the BRISAM users manual is treated separately. The user is referred to that manual for a detailed description of each operation and for general considerations. Only the operations on file types 0, 1 and 2 are presently implemented.

PRECOMPILATION

To maintain the BASIC syntax for BRISAM calls and to attain compatibility with the (not yet released) MBASIC* interpreter interface, a precompiler (BASPCOM) is used to translate BRISAM BASIC program sources to BASCOM sources. All BASIC programs written with BRISAM calls according to this manual must be precompiled using BASPCOM before they can be compiled with BASCOM. For your convenience a default extension of .PBA (prebasic) is assumed by BASPCOM for BRISAM BASIC programs. BASPCOM will read your source file and write an object file with the extension .BAS which can then be compiled with BASCOM.

The precompiler is invoked by the command:

BASPCOM ufr1=ufr2

where ufr1 is the name of the object file to be produced as input for BASCOM and ufr2 is your BRISAM BASIC source code. If no extensions are specified then .PBA is assumed for ufr2 and .BAS is assumed for ufr1. If ufr1 is not specified then the name of the object file will be the same as that of the source file but with the extension .BAS. The equals sign must always be present.

example:

BASPCOM =TEST

reads the BRISAM BASIC program TEST.PBA and writes the BASCOM program TEST.BAS which can then be compiled by BASCOM.

BASPCOM TEST1=TEST2

reads the BRISAM BASIC program TEST2.PBA and writes the BASCOM program TEST1.BAS.

BASPCOM TEST1.BBS=TEST2.BBA

reads the BRISAM BASIC program TEST2.BBA and writes the BASCOM program TEST1.BBS

COMPILATION

Since the precompiler generates some BASIC lines without line numbers the compiler option /C must be used when compiling the BASCOM programs written by BASPCOM.

LINKAGE
The interface routines are contained in the file BASCOMIN.REL which is found on your distribution diskette. This file should be included when linking your object code modules.

example:

```
L80 TEST/N,TEST,BASCOMIN/E
```

links the BASCOM module TEST.REL with BASCOMIN.REL and BASLIB.REL.

NOTATION
All BRISAM BASIC reserved words are shown in boldface and all user supplied entries are shown in normal type. The invocation syntax is shown first followed by a short explanation of the user supplied parameters when not the general case. Make sure that the variable type (string or integer) matches that shown in the examples. A number of reserved words are added to those normally used by BASIC. These are the names of all BRISAM operations as shown in the examples plus the following:

BRDRV$ BRFNA$ BRFNO% BRKL% BRPTR% BRRL% BRTYPE%

The abbreviations used are:

ln an optional line number conforming to BASIC rules

ufr an unambiguous file reference (string) as defined in the BASIC manual for CP/M -- if no drive is specified then the default drive is used

drv a drive reference string such as "A:" or an empty string ("") which specifies the default drive.

fn a BASIC file number which can optionally be written #fn or be an integer variable or expression

all parameters can be expressions of the needed type (string, integer, etc.) or literals except where a variable name is specifically needed as mentioned for the operations BRSETP and BRMOUN.

STATUS
A special operation is included to define which BASIC variable will receive the operation status on operation completion.

ln BRSETP s%

where ln is an optional line number and s% is a user specified BASIC integer variable. After the BRSETP operation has been executed the variable s% will receive the BRISAM status value on the completion of each BRISAM operation until a new "BRSETP" redefines the status variable.

SERIOUS ERROR TRAP

A special operation may be used to trap on serious error conditions (those that return STATUS = 1). Once enabled the trap will be in effect until the program is loaded again or a new program is loaded. After the error message has been displayed the running program is aborted and a return to the operating system is executed.

ln BRTRAP

RECORD BUFFERS AND FIELDS

BRISAM record buffers function the same as BASIC random file record buffers. You can "FIELD" into a BRISAM record buffer in exactly the same way as you would field into a BASIC random file record buffer. Remember that the key is taken from the beginning of the record buffer and can be one or more fields as long as the total key length doesn't exceed 60 bytes. The total record length including the key cannot exceed 252 bytes. You cannot use the same file number for BRISAM files and BASIC files at the same time but once a file is closed the file number can be reused at your discretion. A file BASDUMMY.DAT is created on the default drive so that BASIC has a file it can use for assigning record buffers. This file will always be empty.

"MOUNT VOLUME"

ln **BRMOUN** drv,volname$

The volume on drv is mounted and its volume id is returned in the string variable volname$

"DISMOUNT VOLUME"

ln **BRDISM** drv

"CREATE FILE"

ln **BRCRTE** ufr,rl,kl,type

where rl, kl, and type are the record length, key length and file type respectively. These parameters can be literals or integer expressions.

kl specifies key and sequencer lengths as follows:
file type 0 -- sequencer length * 64 + key length
file type 1 -- secondary key length * 4 + primary key length
file type 2 -- primary key length

"ERASE FILE"

ln **BRERAS** ufr

"OPEN FILE"

ln **BROPEN** fn,ufr,rl

where rl is the record length for the file and must be the same as that specified when the file was created.

"CLOSE FILE"

ln **BRCLOS** fn

the reserved word ALL can be used to close all files instead of specifying one file at a time.

"EMPTY FILE"

ln **BREMPT** fn

"INSERT RECORD"

ln **BRINS** fn

"INSERT RECORD WITH SEQUENCING"

ln **BRINSQ** fn

"DELETE RECORD"

ln **BRDEL** fn

"UPDATE RECORD"

ln BRUPD fn

"GET EQU"

ln BRGEQ fn

"GET GRE EQU"

ln BRGGE fn

"GET GRE"

ln BRGGR fn

"GET LESS EQU"

ln BRGLE fn

"GET LESS"

ln BRGLS fn

"GET USING SECONDARY KEY"

ln BRGUSK fn

"START TRANSACTION"

ln BRBEG

"END TRANSACTION"

ln BREND

"GET ERROR CODE"

ln BRGERR

Note that this operation returns the fatal error code from the last operation that produced a fatal error (status = 1) into the status variable defined at the last BRSETP operation.

