IRIS R8
POLYFILES DOCUMENT

POINT 4
DATA CORPORATION

POINT 4 DATA CORPORATION
2569 McCabe Way / Irvine, California 92714

IRIS R8

POLYFILES DOCUMENT

Revision 07

NOTICE

Every attempt has been made to make this manual complete, accurate and up-to-date. However, all information herein is subject to change due to updates. All inquiries concerning this manual should be directed to POINT 4 Data Corporation.

P R E L I M I N A R Y

REVISION RECORD

PUBLICATION NUMBER: None

Revision	Description	Date
01	Draft Version	
02	Preliminary Version	04/15/82
03	Update incorporating corrections on pages 1-12, 1-14, 1-17, 1-21, 1-31 and 1-32	08/30/82
04	Update to add additional status values to CALL 91 table on page 1-7	10/27/82
05	Update incorporating corrections to pages 1-20, 1-28, 1-31, and 2-2	01/27/83
06	Clarification of the indexed files-to-polyfiles conversion procedure on page 1-31	02/28/83
07	Addition of Section 1.11 (pages 1-34 thru 1-36), Special CALL 91 Modes	06/17/83

LIST OF EFFECTIVE PAGES

Changes, additions, and deletions to information in this manual are indicated by vertical bars in the margins or by a dot near the page number if the entire page is affected. A vertical bar by the page number indicates pagination rather than content has changed.

Page	Rev	Page	Rev	Page	Rev
Cover	-				
Title	07				
ii thru vii	07				
viii,ix	02				
1-1 thru 1-6	02				
1-7	04				
1-8 thru 1-11	02				
1-12	03				
1-13	02				
1-14	03				
1-15,1-16	02				
1-17	03				
1-18,1-19	02				
1-20	05				
1-21	03				
1-22 thru 1-27	02				
1-28	05				
1-29,1-30	02				
1-31	06				
1-32	05				
1-33	02				
1-34 thru 1-36	07				
2-1 thru 2-16	02				
A-1 thru A-7	02				
Comment Sheet	07				
Mailer	-				
Back Cover	-				

PREFACE

The Polyfile is one of the latest enhancements to the IRIS Operating System. It was developed to answer the need for a large data base capability. The best features of the IRIS Contiguous and Indexed file types have been combined, resulting in an extended record capacity per file and allowing for larger sized keys in greater numbers.

This document is an introduction to the use and structure of Polyfiles. It is organized as follows:

Section 1 - a user's guide describing the practical aspects of using and manipulating Polyfiles

Section 2 - general information: theory and structure of Polyfiles, their advantages, and a summary of special Polyfile features.

Appendix A - An exercise in building a Polyfile.

Related manuals include:

Title	Pub. Number
IRIS Installation/Configuration Manual	SM-030-0009
IRIS Operations Manual	SM-030-0010
IRIS 7.3 User Manual	
IRIS Business BASIC Manual	SM-030-0012
IRIS System Commands Manual	SM-030-0011

CONTENTS

FIGURES

TABLES

SECTION 1

This section is a guide to the use of Polyfiles. The various sections are devoted to the creation, use, and manipulation of Polyfiles.

1.1 INTRODUCTION

A Polyfile is made up of many files. Each file is a contiguous type file and is called a volume. Each volume may reside on a different Logical Unit (LU) and the various volumes are tied together into one entity by the filename, access date, and a master volume.

The master volume is always volume 0 and it contains information about all the other volumes making up a Polyfile. The various types of volumes are:

The Master Volume
Base Directory Volume
Directory Extension Volumes
Data Volumes
Unstructured Volumes

Backups must be performed frequently and on a regular basis to preserve the integrity of Polyfiles. Please refer to the IRIS Operations Manual for recommended backup procedures.

Three BASIC programs have been written to aid you in the construction and manipulation of Polyfiles. They are BUILDPF, QUERYPF, and KILLPF. All three make use of the $TERMS mnemonics.

There are some important points to remember when building a Polyfile. These are:

- A Polyfile must have a unique name
- A Polyfile may not be built on LU/0. The various volumes may reside on LUs 1 - 127
- Polyfiles may have Data Volumes that are mapped but, in any one Polyfile, the Data Volumes must be either mapped or unmapped
- Maximum number of volumes in a Polyfile is 64 (volumes 0 - 63). The size of a volume is limited by the size of the LU on which it resides (Maximum size = 65335 records including the header and map blocks)
- Record size must be uniform throughout a Polyfile. The first record in a Data Volume is numbered 0. Records may extend over block boundaries.
- Keys in the Directory Volumes (Base or Extended) may be up to 121 bytes in length.

For a more extensive discussion on Polyfile structure please see Section 2.

1.2 POLYFILE NAMES

Polyfile names are subject to the same conventions as other IRIS files. A name is a string of upper or lower case characters which may consist of alpha characters, numerics, and periods. A Polyfile name includes an LU number and a trailing @-sign. The maximum number of characters allowed is 13 plus @nn (i.e. a total string of up to 16 characters). However, the 13 characters before the @-sign must include the optional password and the <CTRL-E> codes associated with the password. The two characters after the @-sign may be a volume number or some other identifier. If an exclamation mark is used for read only access (see Section 1.4.3) then '!' is considered part of the two characters following the @-sign.

Examples of acceptable Polyfile names are:

```
    PFname<CTRL-E>restr<CTRL-E>@d
    pf.Name1@30
    Pf2name@4!
```

If a polyfile is created as contiguous files, the trailing @-sign is added automatically when these separate files are structured into a Polyfile (see Section 1.3).

When a Polyfile is built using the BUILDPF utility (see Section 1.3.2), then the format for its name is:

```
    n/PF.name@
```

The name of a Polyfile should be unique. It should be different from any filename existing on those LUs on which the Polyfile volumes may be built.

For Example: If a file with the name ABC exists on LU/3 and a volume of your Polyfile is to be built on LU/3, then you may _not_ call the Polyfile ABC@. The @-sign does not make the file name unique and the name ABC@ will be rejected by the system.

1.3 BUILDING POLYFILES

There are two ways in which a Polyfile may be built. You may build a Polyfile by first building a number of Contiguous files (see Section 1.3.1 for the procedure).

The recommended way for building a Polyfile is to use the BUILDPF utility (see Section 1.3.2).

1.3.1 BUILDING A POLYFILE FROM CONTIGUOUS FILES

Three steps are involved in creating each Polyfile volume when the BUILDPF utility program is not used. They are:

1. Build a contiguous file with the polyfile name less the "@" by using the FORMAT processor or the BASIC "BUILD" statement.

2. Transform the contiguous file into a polyfile volume via the polyfile call (CALL 91). The V parameter in CALL 91 assigns the volume numbers.

   NOTE

   Volume 0, the Master Volume, must be built first.

3. Structure the polyfile volume via Search Mode 0.

1.3.1.1 Step 1 - Build a Contiguous File

The first step in building a polyfile is to BUILD a contiguous file. For example:

```
BUILD #C0,"[R:L] $ddd.cc <pp> LU/Polyfilename"
```

where:

C0	- Channel Number
R	- Number of data records to be in this volume
L	- Length of a data record in words
ddd	- Dollars charged for file access
cc	- Cents charged for file access
pp	- Protection level
LU	- Logical Unit on which to build the volume

1.3.1.2 Step 2 - Convert to Polyfile Volume

The contiguous file is then converted to a Polyfile volume by a CALL command (see below) which must be executed while the file is still in the BUILD mode. After the call is executed, channel C0 must be closed to make the volume permanent on the disc. When building the master volume (0), channel C1 should be closed. Enter the following statements:

```
    IF ERR 0 STOP
    CALL C,C0,C1,V,S,P
```

where:

```
    C  - CALL number (91)
    C0 - Channel number on which the file is open
    C1 - Channel number on which the master volume is open
    V  - Volume number
    S  - Status after the CALL is completed
    P  - File parameter array (see Table 1-1)
```

NOTE

> The variable S and the array P must be declared in a DIM statement; otherwise, the 'IF ERR' branch will take effect and S will return error 17, 18, or 19 (see Table 1-2). The format for the DIM statement is:
>
> ```
>     1234 DIM S,P[n]
> ```
>
> where n is a value of 10 or greater. This permits validation of the P array dimensions.

- If CALL 91 detects any errors, an error #38 (error detected by a called subroutine) will be returned.
- CALL C checks to see that the Filename matches and that the volume is three or more blocks in size.
- If C0 is less than 0, then only file parameters are returned.
- If C0 is greater than or equal to 0, and if

  V>0 - this volume is linked to volume 0.
  V=0 - a master volume is created.
  V<0 - CALL C assigns the next available volume number and returns the number of that volume in V.

- If the volume is a data volume, then the record length of the volume must match that of the master volume.

- When C0 is non-negative, S should be set:

  S=0 - Volume is to be a Base Directory or Directory Extension

  S<>0 - Volume is to be a Data Volume

- If S is returned from the call with value 0, then file parameters are to be found in the array P. The parameters, ordered by index are shown in Table 1-1.

TABLE 1-1. FILE PARAMETERS

Contents of P	Description
0	VLU (Volume/Logical Unit)
1	BNR (Base Volume/Logical Unit)
2	LU flag (0=installed; <>0=not installed)
3	ACNT
4	TYPE
5	NBLK
6	LRCD
7	NRCD
8	LDAT
9	LDAT+1
10	Keylength of 1st directory in volume (FMAP+4)
11	Keylength of 2nd directory in volume (FMAP+5)
:	
72	Keylength of 63rd directory in volume (FMAP+102)
76	Logical Unit number
77	DHDR

If S is returned nonzero, then an error is indicated. The possible status values for S are shown in Table 1-2.

TABLE 1-2. CALL 91 STATUS VALUES

Contents of S	Description
0	No error
1	Invalid channel number
2	File not being built
3	Illegal volume number
4	File C1 is not a polyfile
5	File name invalid
6	Invalid variable type
7	Invalid number
8	Volume already exists on the desired LU, possibly as part of another polyfile of the same name
9	File C0 not found (deleted?)
10	Not enough nodes to link into extended DFT
11	Volume already exists for this polyfile
12	Volume V not found
13	Account numbers do not match
14	Volume in extended DFT but not on disc
15	No available volume number for this polyfile
16	Volume V is not defined
17	P is not allocated as the next variable after S
18	P is not an array
19	P is not dimensioned P[10] or greater
20	File C0 is not contiguous
21	File C1 is open elsewhere
22	File C0 is already a polyfile
23	HSLAS do not match for assign operations
24	VLU or BNR do not match for assign operations
25	Cannot move volume 0
26	Illegal move operation
27	File C0 is not write protected
28	Illegal write enable operation

1.3.1.3 Step 3 - Structure Polyfile

Structuring a polyfile is done with SEARCH mode 0 in BASIC. However, in the case of polyfiles, SEARCH mode 0 requires that a volume number be given and that key sizes be given in bytes. The format is:

```
    SEARCH #C,m,d;v$,v1,v2
```

where:

m=0,d+128	- Define volume number d to be a data volume. 0<=d<=63, v1=number of records. • If v1=0, the number of records is computed and returned in v1. • If v2=0, a free record map is not built. • If v2<>0, a free record map is built with v1 free records in it.
m=0,d+64	- Define volume number d to be an extension of the base directory volume in v2. 0<=d<=63.
m=0,d	- Define directory d: volume number in v2, key length in v1 (in bytes) 1<=d<=63.
m=0,d=0	- Organize all directories for the volume number given in v2.

1.3.2 BUILDPF

BUILDPF is a utility written in BASIC to assist the user in the creation or extension of a polyfile. A BUILDPF exercise is provided in Appendix A which steps through all the different commands discussed in this section.

All responses to the system prompts require a <RETURN> for them to be "entered".

At the system prompt (#) enter

```
    BUILDPF
```

The system responds

```
    BUILDPF - Build Polyfiles Utility
```

It then requests a filename with the prompt:

```
    POLYFILENAME [must have "LU/" (not 0)]:
```

A Polyfile must not be built on LU/0. The range of possible LUs is from 1 to 127.

The name must be terminated by an "@" as with all polyfile names.

BUILDPF then attempts to open the file. If the polyfile is found, then BUILDPF enters polyfile extension mode (see Section 1.3.2.4). If the file is not found, BUILDPF prompts

```
    POLYFILE NOT FOUND
    DO YOU WISH TO CREATE A NEW ONE? (Y/N)
```

If the answer is N (no), then the user is returned to SCOPE. If the answer is Y (yes), BUILDPF requests a record size:

```
    RECORD SIZE (in words for the entire Polyfile):
```

- Possible record sizes range from 1 to 16383 words (2^{14} - 1).
- A record size must be given even if the polyfile will not contain Data Volumes.
- The record size of both Base Directory volumes and Directory Extension volumes should be 256 words. This means that the size of these types of volumes is limited only by the available space on an LU.

  Volume 0 is an exception to this, it must always have the record length which is declared for the entire polyfile regardless of volume type. A polyfile record length of less than 256 words will restrict volume 0 to:

  Size (in blocks) = (65535 * (record size) / 256)

After a record size has been entered, BUILDPF will print the following message:

```
    VOLUME: 0
```

This output is to remind the user that he is entering parameters for volume 0, the master volume. Next BUILDPF will request volume type information. For details on further prompts and responses, see Section 1.3.2.1.

1.3.2.1 Volume Types Input

When BUILDPF requests a volume type, the prompt is:

```
    VOLUME TYPES:  "B"  Base Directory
                   "E"  Extension Directory
                   "D"  Data Volume
    VOLUME TYPE:
```

1.3.2.1.1 BASE DIRECTORY

If type "B", BUILDPF prompts:

```
    STARTING DIRECTORY NUMBER FOR THIS VOLUME:
```

The request is for the lowest numbered directory to be based in the volume. The specified directory number must be in the range 1 to 63 and must not be defined in an existing base directory volume.

BUILDPF then displays:

```
    DIRECTORY NUMBERS AVAILABLE FROM N0 THRU N1
```

"N0" is the specified starting directory number and the range "N0 thru N1" is the range of contiguously available directory numbers starting with N0. BUILDPF will give the following prompt for each of the volume numbers in the range N0 - N1:

```
    DIRECTORY xx KEY SIZE IN CHARACTERS [<RETURN> TO TERMINATE]:
```

Valid key sizes range from 2 to 121. Pressing <RETURN> only will terminate the definition of directories. BUILDPF then prompts:

```
    THIS DIRECTORY SETUP OK?
```

Input of <RETURN> only, "Y", or "y" approves the directory setup. Any other input causes a return to the prompt for directory number N0.

1.3.2.1.2 EXTENSION DIRECTORY

If type "E", then BUILDPF prompts:

```
    VOLUME TO EXTEND:
```

The volume entered must exist and must be a base directory volume.

1.3.2.1.3 DATA VOLUME

If type "D", BUILDPF prompts:

```
    DATA VOLUME(S) TO HAVE MAPS? ["Y"/"N"]:
```

This prompt appears once only because the answer given to this question will then apply to all data volumes in the polyfile.

| Refer to Section 2.2.2.2 for a discussion on data volume maps.

1.3.2.2 Volume Size Input

BUILDPF next requests volume size information. This may be input in three general forms: number of records, number of keys, or number of disc blocks.

There are two types of valid size information for Base Directory Volumes or Directory Extension Volumes:

Number of keys
Number of disc blocks (less header)

1.3.2.2.1 BASE DIRECTORY VOLUME

A base directory volume will cause the following prompt:

```
VOLUME SIZE IN INDEXES (KEYS)
[NEGATIVE FOR SIZE IN BLOCKS]:
```

If the volume size is given in keys, the size of the volume is computed using the following assumptions:

1. The number of keys specified is the maximum number of keys across all directories of the base volume.

2. The key size used for volume size computation is the largest key size defined in the directories of the base directory volume.

3. The size of the fine and intermediate directory levels is based on a fullness factor of .5 which is stored in the variable K9. K9 is defined early in the program and may be adjusted by the user if the need arises.

1.3.2.2.2 DIRECTORY EXTENSION VOLUME

A directory extension volume causes the following prompt:

```
BASE VOLUME BB AND ITS CURRENT EXTENSIONS HAVE
A TOTAL OF NNN BLOCKS WHICH HOLD A MAXIMUM OF
APPROXIMATELY KKKK KEYS.

NEW MAXIMUM NUMBER OF INDEXES (KEYS)
[NEGATIVE FOR SIZE IN BLOCKS]:
```

The input, representing a new maximum number of keys, must have a value greater than kkkk. The new value is used to compute the total number of blocks needed to hold the keys. The difference between the new computed total blocks and kkkk is used to determine the size of the new volume. If the size is given in blocks, the size applies only to the volume being defined.

1.3.2.2.3 DATA VOLUME

The size of the volume must be stipulated at this point. It may be given in total number of records or total number of blocks required. BUILDPF requests input as follows:

```
VOLUME SIZE IN RECORDS
[NEGATIVE FOR SIZE IN BLOCKS]:
```

If a positive number is entered, BUILDPF assumes the total number of records was stipulated. Based on the record size entered previously and the total number of records entered here, BUILDPF calculates the number of blocks required for the volume.

If a negative number is entered, then BUILDPF accepts this as the number of BUILDPF blocks required for this volume.

1.3.2.3 BUILDing and Structuring Volumes

The building and structuring phase of polyfiles is the critical phase of BUILDPF. If an <ESC> or <CTRL-C> is pressed while theprogram is in this phase, a misformed polyfile may result. The beginning of the phase is indicated by the message:

```
ALLOCATING VOLUME.  PLEASE WAIT.
```

When allocation is complete, BUILDPF displays:

```
VOLUME xx ALLOCATION COMPLETE.
```

Structuring then takes place indicated by one of the following messages:

```
STRUCTURING VOLUME xx AS BASE DIRECTORY VOLUME. PLEASE WAIT.

STRUCTURING VOLUME xx AS A DIRECTORY EXTENSION. PLEASE WAIT.

STRUCTURING VOLUME xx AS A DATA VOLUME.  PLEASE WAIT.
```

After a base directory volume or directory extension volume has been built and structured, the following prompt will appear:

```
EXTEND BASE VOLUME bb MORE ["Y"/"N"; <RETURN> = exit]:
```

A "Y" response causes BUILDPF to assume volume type "E" (directory extension) and base volume bb (see Section 1.3.1.4 for Volume Types Input). An "N" response returns to the initial prompts for a Logical Unit and the opportunity to build another type volume.

When structuring is complete, BUILDPF displays the messages:

```
STRUCTURING COMPLETE.

LOGICAL UNIT (NONZERO) FOR VOLUME [<RETURN> = EXIT] :
```

This last prompt gives you the opportunity either to continue with building other volumes or to exit the BUILDPF program.

At this point it is again safe to use <ESC> or <CTRL-C>.

1.3.2.4 Polyfile Extension Mode

After volume 0 has been structured, BUILDPF enters extension mode. If the polyfile is found, then BUILDPF prints the message:

```
    FILE FOUND.  POLYFILE EXTENSION MODE.
```

You are now in polyfile extension mode. You may now add volumes to the polyfile or structure ones which for some reason are unstructured.

BUILDPF then prompts:

```
    LOGICAL UNIT (NONZERO) FOR VOLUME  [<RETURN> = EXIT]:
```

The LU number input must be in the range 1 to 127. Pressing <RETURN> only causes the program to exit to SCOPE. BUILDPF next prompts

```
    VOLUME NUMBER [0-63; <RETURN> = don't care]:
```

BUILDPF is asking for the volume number of the new volume to be created or the volume number of an existing but unstructured volume.

The recommended input is <RETURN>, to allow BUILDPF to choose the lowest available volume number for the selected volume type.

After the volume number is input, BUILDPF will request volume type and volume size before building and structuring the volume.

1.4 ACCESSING POLYFILES

The opening of a Polyfile for a BASIC user is the same as the opening of any other IRIS file type but three extra conditions must be met:

1. The Master Volume (Volume 0) must reside on the Logical Unit (LU) specified in the OPEN statement. The default (if no LU is specified) is the user's assigned LU.
2. All the LUs where the volumes of a Polyfile reside must be installed.
3. Hours Since Last Access (HSLA) of all the volumes must be identical to ensure Polyfile integrity.

NOTE

HSLA may be violated because of a faulty backup or backdown.

1.4.1 OPEN A POLYFILE

For a BASIC user, a Polyfile is opened with the following command:

```
OPEN c#,Pfname@
```

When a Polyfile is opened, the HSLA (Hours Since Last Access) is updated for all the volumes in the Polyfile. The value of HSLA must always be the same for all the volumes contained in a Polyfile.

A single volume in a Polyfile may be opened by stipulating the volume number as follows:

```
OPEN c#,lu/Pfname@vo
```

where lu is the Logical Unit on which the volume resides and vo is the volume number.

Adding an exclamation mark (!) after the @-sign or the volume number permits the opening of a polyfile or an individual volume in a polyfile with a "read-only" status (see Section 1.4.3).

A polyfile name with an exclamation mark added does not overlay the original file.

1.4.2 READ A POLYFILE

The READ command is used when a record in a Polyfile is to be accessed by the record number. For key access see Section 1.4.4.

The format for the READ statement is as follows:

```
    READ #c,r,o;v1,v2,...;
```

```
where c - channel number
      r - record number
      o - byte offset from the beginning of the record
      v1 and v2, etc. are numeric or string variables
```

NOTE

> In a Polyfile, data record numbers start at zero.

For a detailed discussion on the READ statement, please see the IRIS User Manual (1978), Sections 12.7 and 12.8.

1.4.3 READ ONLY ACCESS

There may be times when it is desirable to examine the data in a Polyfile volume. This can be done by adding an exclamation mark (!) to the name assigned to individual Polyfile volumes. For example:

```
    OPEN #1, "PFFILE@5!"
```

will open PFFILE volume number 5 on channel 1 in a write-locked (read only) condition.

The HSLA is not updated because the file integrity is not violated by a read-only access.

1.4.4 WRITE STATEMENT

The format for the WRITE statement is similar to the read statement. For example:

```
    WRITE #c,r,o;v1,v2,....:
```

```
where c - Channel number
      r - Record number
      o - byte offset from the beginning of the record
      v1 and v2 are numeric or string variables
```

For more information see Section 12.8 of the IRIS User Manual (1978).

1.4.5 SEARCH STATEMENT

The SEARCH statement may be used by the BASIC user in two ways:

1. To structure a Polyfile
2. To access a Polyfile by key

Search mode 0, used for structuring a Polyfile, is discussed in Section 1.3. Search modes used for accessing a Polyfile by key include:

SEARCH mode 1 - Return key length in bytes

m=1,d>0 - Reads key length in bytes of directory d into v1. If d directory does not exist then v1 is not changed and v2 is changed to 13.

m=1,d=0 - Miscellaneous functions that return values similar to indexed files. The function is specified in v2:

v2=0 - First real record number on indexed files. This is always zero for a Polyfile.

v2=1 - Number of free records. This is the sum of all available records across all Data Volumes in the Polyfile.

v2=2 - Allocate and return record number of the first available free record. v2=3 if there is no free record to be allocated.

v2=3 - Releases record v1. v2=19 if the record was already free. v2=20 if the Polyfile does not have that record number.

SEARCH mode 2 - Looks for a matching key and returns the associated record number

SEARCH mode 3 - Searches for the next highest key and returns it with the associated record number

SEARCH mode 4 - Inserts a new key

SEARCH mode 5 - Deletes a key

SEARCH mode 6 - Searches for next lowest key and returns the associated record number. If there are no keys less than v$ then the value in v2 is set to 2. If there is a lower key, set the value in v2 to 1. (To be implemented).

SEARCH mode 7 - Not in use

SEARCH mode 8 - Places the next used record number greater than v1 in v1 (to be implemented)

For BASIC users the format of this type of SEARCH command is:

```
    SEARCH #c,m,d;v$,v1,v2
```

where
- c - channel number
- m - SEARCH mode
- d - directory volume number
- v1 - number variable; receives the result of the search, i.e. the record number
- v2 - number variable; receives the status of the search.

These values are shown in Table 1-3.

TABLE 1-3. V2 STATUS VALUES

Status	Description
0	No error, operation successful
1	Operation not successful
2	End of directory (on insert, indicates directory is full)
3	End of data (no free records available)
4	File not indexed
5	Polyfile structure error
6	Directory number not in sequence
7	File is not contiguous
8	Volume is already indexed
9	Illegal key length (less than 2 or greater than 121)
10	Too many directories (limit is 63 per volume/polyfile)
11	Volume not found (possible structure error)
12	Volume too small
13	Directory not found
14	File not indexed
15	Data volume number is less than pre-existing data volume
16	Data volume map request not consistent with pre-existing volumes
17	Data volume does not have record length matching that of the polyfile
18	Block/record out of range
19	Record was not allocated (already released)
20	Volume has no map

1.4.6 CLOSE STATEMENT

The format for the CLOSE statement is:

```
    CLOSE #c
```

where c is the channel number.

1.5 QUERYPF

QUERYPF is a system utility which generates information about the status of existing Polyfiles. The information can be generated in three forms:

- Individual Volume Display
- Global Display
- Informational Dump

The output generated may go to the terminal, a file, or a device such as $LPT.

At the system prompt (#), enter:

```
    QUERYPF
```

The system responds

```
    QUERYPF - Query Polyfiles Utility

    Polyfile name [should have LU/]:
```

The polyfile master volume (0) should be present on the specified logical unit. If no LU is specified, the master volume must be found on the default (assigned) logical unit.

Please note that the polyfile name input here must have a trailing "@" as must all polyfile names.

After the polyfile master volume has been found, QUERYPF prompts:

```
    Output file [<RETURN> = output to terminal]:
```

Carriage return (<RETURN>) alone causes output to go to the terminal. A file name or a device name may be specified for the output. After an appropriate response, there is a brief pause while the program gathers file information to display. The next prompt is:

```
    Please input volume number [0-63]
         or <RETURN> for global display
         or      -1 for complete dump
         or ESCape to exit to SCOPE :
```

1.5.1 INDIVIDUAL VOLUME DISPLAY

Input of a volume number causes display of volume characteristics if the volume exists. For example:

```
Volume: 0       DHDR:  1/001130          Logical unit   1 installed.
Privilege level: 2      Group: 1  User 4    Protection: 77
Size: 23 disc blocks
Volume is a Base Directory volume with 3 directories.
Base Volume 0 and its current extensions have a total of 73 blocks
for keys which will hold a maximum of approximately 694 keys.
Directory:  Key length (in characters)
       7: 10     8: 31     9: 25
```

1.5.2 POLYFILE GLOBAL DISPLAY

If <RETURN> is pressed, global information for the file is displayed:

```
DEC 28, 1981  12:14:29
Polyfile: 1/DEMOFILE@
Volume 0 LU: 1               Total data records allocation: 589
              Total data records allocation: 614
Record size is 20 words  Total volumes: 7  Last accessed:  0.09 hours ago

Vol LU/ NBLK Type | Vol LU/ NBLK Type | Vol LU/ NBLK Type | Vol LU/ NBLK Type
 0   1/  43   B 7 |  1   1/  19   E 0 |  2   1/  42   D M |  3   1/  89   B 1
 4   1/ 100   B25 |  5   1/  25   E 0 |  9   1/  11   D M |
```

The table above is largely self-explanatory but the type column needs explanation. The type column may contain any of the following:

Bnn	Base directory volume whose lowest directory number is nn.
Ebb	Directory extension volume whose base volume is bb
D M	Data volume. If the "M" is present, the volume has a map.
U	Unstructured volume. Use BUILDPF to structure it.
P**	Partially structured file. This can only occur from a disaster of some sort when building a polyfile.
***	Illegal volume type.

If the volume number is followed by an asterisk (*), that volume will prevent the polyfile from opening. Possible reasons for the flag are:

1. The HSLA does not match the master volume.
2. Volume was not found on the LU specified by the master volume.
3. Volume's LU not installed.
4. Account number does not match master volume.
5. Volume is unstructured.

1.5.3 COMPLETE DUMP

An input of a "-1" causes a global display output followed by individual volume information for each volume in the polyfile. In the following example, volume 4's LU is not installed. Note that some of the information in the individual volume display for volume 4 has no meaning.

```
DEC 28, 1981  12:14:29
Polyfile: 1/DEMOFILE@
Volume 0 LU: 1                Total data records allocation: 589
             Total data records allocation: 614
Record size is 20 words  Total volumes: 7  Last accessed:  0.09 hours ago
Vol LU/ NBLK Type | Vol LU/ NBLK Type | Vol LU/ NBLK Type | Vol LU/ NBLK Type
 0   1/  43   B 7 |  1   1/  19   E 0 |  2   1/  42   D M |  3   1/  89   B 1
 4   1/ 100   B44 |  5   1/  25   E 0 |  9   1/  11   D M |


Volume: 0      DHDR:  1/014060          Logical unit   1 installed.
Privilege level: 2      Group: 1  User 4    Protection: 77
Size: 43 disc blocks
Volume is a Base Directory volume with 3 directories.
Base Volume 0 and its current extensions have a total of 81 blocks
for keys which will hold a maximum of approximately 486 keys.
Directory: Key length (in characters)
      7: 10     8: 31     9: 25


Volume: 1      DHDR:  1/013661          Logical unit   1 installed.
Privilege level: 2      Group: 1  User 4    Protection: 77
Size: 19 disc blocks
Volume is a Directory Extension of volume 0.


Volume: 2      DHDR:  1/014133          Logical unit   1 installed.
Privilege level: 2      Group: 1  User 4    Protection: 77
Size: 42 disc blocks
Volume is a Data volume which holds 524 records.
Volume is mapped.  Map size is 1 blocks.


Volume: 3      DHDR:  1/014205          Logical unit   1 installed.
Privilege level: 2      Group: 1  User 4    Protection: 77
Size: 58 disc blocks
Volume is a Base Directory volume with 4 directories.
Base Volume 3 and its current extensions have a total of 87 blocks
for keys which will hold a maximum of approximately 1001 keys.
Directory: Key length (in characters)
      1: 17     2: 12     3:  7     4:  5
```

If a logical unit given in the building process was not installed (in this case LU/2), then an asterisk (*) will appear next to the volume and the informational display will be meaningless. An example of the global display follows (the individual volume information would be the same as in the previous example):

```
 Vol LU/ NBLK Type | Vol LU/ NBLK Type | Vol LU/ NBLK Type | Vol LU/ NBLK Type
  0   1/  43   B 7 |  1   1/  19   E 0 |  2   1/  42   D M |  3   1/  89   B 1
  4*  2/       B44 |  5   1/  25   E 0 |  9   1/  11   D M |
```

If LU 2 was not installed then the display for volume 4 will be as shown below. Notice that the information given does not make any sense.

```
Volume: 4      DHDR:  2/??????        ** Logical unit    2 NOT installed. **
Volume is a Base Directory volume with 1 directories.
Base Volume 4 and its current extensions have a total of-2 blocks for keys
which will hold a maximum of approximately-125 keys.
Directory: Key length (in characters)
     44:  0
```

If LU 2 was installed, then the display for volume 4 will be as follows:

```
Volume: 4      DHDR:  2/000043         Logical unit    2 installed.
Privilege level: 2      Group: 1  User 4    Protection: 77
Size: 100 disc blocks
Volume is a Base Directory volume with 4 directories.
Base Volume 4 and its current extensions have a total of 98 blocks
for keys which will hold a maximum of approximately 98 keys.
Directory: Key length (in characters)
      44:120


Volume: 5      DHDR:  1/014336         Logical unit    1 installed.
Privilege level: 2      Group: 1  User 4    Protection: 77
Size: 25 disc blocks
Volume is a Directory Extension of volume 0.


Volume: 9      DHDR:  1/013704         Logical unit    1 installed.
Privilege level: 2      Group: 1  User 4    Protection: 77
Size: 11 disc blocks
Volume is a Data  Volume which holds 116 records.
Volume is mapped.  Map size is 1 blocks.
```

When all the volumes have been displayed, the initial prompt is repeated to give you a choice of redisplaying it or exiting:

```
Please input volume number [0-63]
     or <RETURN> for global display
     or      -1 for complete dump
     or  ESCape to exit to SCOPE:
```

1.6 BUILDPFERR

Both BUILDPF and QUERYPF attempt to open the file 0/POLYFILERRORS. This file contains error messages for both processors. If the file is not found, then error codes are output, and must be looked up by the user. To create 0/POLYFILERRORS, the program BUILDPFERR should be run from the utility account (0,2).

Table 1-4 gives the list of CALL 91 errors. Table 1-5 lists the possible search mode statuses contained in the BUILDPFERR file. Both tables give the associated file record numbers.

TABLE 1-4. CALL 91 ERROR LIST

CALL 91 Error	File Rec.#	Description
1	1	Bad Channel #
2	2	File not being built
3	3	Bad Volume #
4	4	C1 file not polyfile
5	5	Bad filename
6	6	Bad var type
7	7	Bad number
8	8	Volume pre-exists on LU. May be part of another polyfile or fragment
9	9	File C0 not found
10	10	No nodes
11	11	Volume already defined for this polyfile
12	12	Not used
13	13	Account numbers differ
14	14	Vol in Data File Table (DFT) but not on disc
15	15	No available vol numbers
16	16	Volume not defined. It is missing
17	17	P does not immediately succeed S in VDT
18	18	P is not an array
19	19	P is not dimensioned at least P[10]

TABLE 1-5. SEARCH MODE STATUS LIST

Status	Rec.#	Description
0	100	No error, operation successful
1	101	Operation not successful
2	102	End of directory (on insert, indicates directory is full)
3	103	End of data (no free records available)
4	104	File not indexed
5	105	Polyfile structure error
6	106	Directory number not in sequence
7	107	File is not contiguous
8	108	Volume is already indexed
9	109	Illegal key length (less than 2 or greater than 121)
10	110	Too many directories (limit is 63 per volume polyfile)
11	111	Volume not found (possible structural error)
12	112	Volume (built) too small
13	113	Directory not found
14	114	File not indexed
15	115	Data volume number is less than the preexisting data volume
16	116	Data volume map request not consistent with preexisting volumes
17	117	Data volume does not have record length matching that of the polyfile
18	118	Block record out of range
19	119	Record was not allocated (already released)
20	120	Volume has no map

1.7 KILLPF

KILLPF is a system utility specially designed to delete Polyfiles. You can use the KILL processor but then you must KILL each volume in a Polyfile separately and you must make sure that Volume 0 is deleted last.

KILLPF will report on each volume as it is deleted. The procedure is as follows:

At the system prompt, enter:

```
KILLPF
```

The system responds by acknowledging the command and asking for the Polyfile name:

```
KILLPF - Kill Polyfile Utility

Polyfile name [should have "LU/"]:
```

Enter the Polyfile name; the display will look something like this:

```
Polyfile name [should have "LU/"]: 1/PFname@ Validating polyfile structure.
Please wait.
March 16, 1982  21:55:10
Polyfile:  PFname@
Volume 0 LU: 1  Total data records allocation: 0
Record size is 256 words Total volumes: 3 Last accessed: 0.46 hours ago
Vol LU/ NBLK Type | Vol LU/ NBLK Type | Vol LU/ NBLK Type | Vol LU/ NBLK Type
 0   1/  25   B 1 |  1   1/  24   E 0 |  2   1/  42   D M |  3

Type "YES" to confirm deletion:
```

After you have typed in the full word "YES" and pressed <RETURN>, each volume will be listed as it is deleted:

```
Volume 2  deleted
Volume 1  deleted
Volume 0  deleted
Polyfile deletion sequence complete
#
```

At the end of the sequence you are returned to the normal system prompt.

NOTE

> If a volume is marked by an asterisk (i.e., the LU is not installed), the prompt to confirm deletion will be repeated. Before the delete process can continue, the LU must be installed.

1.8 CONVERTING BASIC PROGRAMS - BCONVERT

The BCONVERT processor provides an easy way to convert standard R7 BASIC programs to the R8 BASIC program format. Program sizes will increase slightly but with the capability of dynamic partitioning under IRIS R8, this presents no problem.

BCONVERT may be invoked by any user but it will convert only those programs that are on the user's account.

When BCONVERT has completed its operation, it chains to BASIC. You may then list the converted program if it was not PROTECTed.

If you wish to save the program, then exit to SCOPE with the EXIT command and invoke the SAVE processor.

A sample dialogue using BCONVERT is shown below. The user input is underlined.

```
    #BCONVERT R7PROGRAM  --  Converting R7.x to R8.0 format
    EXIT
    #SAVE R8PROGRAM
    SAVED !!    CHECK CODE = ACDX
    #
```

1.9 CONVERTING INDEXED FILES TO POLYFILES

Two major phases are involved in converting an Indexed file to a Polyfile: file conversion and program conversion. The steps for each phase are shown below:

A. File Conversion

1. Build a Polyfile with key and data record sizes matching the current Indexed File.

2. Construct and execute a program which will

   a. Read each key and its associated data from the Indexed File

   b. Get a free record from the Polyfile

   c. Write the data record into the Polyfile

   d. Insert the key into the Polyfile

   e. Allow at least a 3% (precision) variable for the Polyfile record number

   f. Use separate variables for Indexed File access and Polyfile access to prevent V1 and V2 being greater than 65535 for Indexed File access

B. Program Conversion

1. Modify OPEN statements to reference the Polyfile.

2. Remove Search Mode 7s from the program. Polyfiles have the capacity to redistribute keys automatically.

NOTE

Search Mode 7 has no effect on Polyfiles. A program containing Search Mode 7 may safely be used; however, Search Mode 7 is unnecessary and should be eliminated.

When used with an Indexed File, Search Mode 7 usually (but not always) allows more keys to be inserted after a V2=2 (directory full) is encountered. For Polyfiles, Search Mode 7 has no effect on getting a V2=2. Programs which assume that Search Mode 7 always allows more keys to be inserted after V2=2 are incorrect for both Indexed files and Polyfiles and should be changed.

3. A variable intended for Polyfile record numbers should be large enough to hold 3% (precision) numbers.

1.10 USING OTHER SYSTEM PROCESSORS

In general, it will not be necessary to use other system commands to manipulate or list a Polyfile. This section briefly describes the commands that may be used.

1.10.1 CHANGE

If you wish to change the name of a Polyfile you must change the name on each volume. Therefore, run the QUERYPF utility and get a global display to make sure that you have a list of all the volumes in that Polyfile and their associated LU numbers.

The Polyfile must not be in use while its name is being changed.

File access protection for a Polyfile is the protection given to the Master Volume, the other volumes should have the default protection level of 77.

Invoke the CHANGE processor giving the LU number, Polyfilename@, and the volume number. See also Section 2.2 of the IRIS User Manual (1978).

1.10.2 QUERY

The system processor 'QUERY' may be used on individual volumes of a Polyfile. Each volume will appear as a simple contiguous file. To get Polyfile type information, use QUERYPF.

1.10.3 LIBR

If LIBR is used, each volume of a Polyfile will appear as an individual file in the LIBR display as follows:

```
    Filename@xx
```

where xx may be a two-digit number, or it may be zero.

1.10.4 INSTALL

If a LU number is changed during an INSTALL, the following warning message is given:

```
WARNING CHANGING LOGICAL UNIT NUMBER MAY INVALIDATE POLYFILE
VOLUME Pfname@
```

A Polyfile with a volume on the changed LU will not OPEN because the information in the Master Volume is now incorrect and the system cannot find the volume.

When the LU is reset to its original number, the Polyfile will open.

1.11 SPECIAL CALL 91 MODES

CALL 91 is the polyfile call that may be used to link a new polyfile volume from a BASIC program (as discussed in Sections 1.3.1 through 1.3.1.3). Three special CALL 91 modes have been added to the basic CALL 91. They are used primarily by the ASSIGNPF and COPYPF programs.

- ASSIGNPF is used to reassign the number of a logical unit which has one or more polyfile volumes resident. The program must be run if logical unit numbers have been changed during an INSTALL procedure (see the IRIS Operations Manual).

- COPYPF is used to move polyfiles from one logical unit to another. This procedure is discussed in the IRIS System Commands Manual.

All requirements for the normal CALL 91 apply to the three special modes.

1.11.1 CALL 91 - SPECIAL MODE 1

Allows an individual volume of an existing polyfile to be moved (copied) to another logical unit. The original (source) volume remains on the source logical unit. The new volume must first be built on the destination logical unit without being closed. The new volume must have the same filename as the original polyfile (i.e., all the characters preceding the @-sign must be the same). That name must not already be in use on the destination logical unit. CALL 91 special mode 1 is then used to link the new volume to the existing polyfile.

The syntax of a CALL 91 special mode 1 statement is as follows:

```
    CALL 91,C0,-C1,V,S,P
```

where

C0	-	Channel on which the new volume is open with build bit set (i.e., it must not have been closed)
-C1	-	Channel on which the entire polyfile is open*
V	-	Number of the new volume which must be the same as the source volume
S	-	Status to be returned (S=0 if no error 38)
P	-	Array in which file parameters are returned (see Table 1-1)

To preserve polyfile integrity, CALL 91 will cause an error 38 and return S=22 if the polyfile is open by another user.

If the call is successful, the entire polyfile remains open-locked until the channel is closed.

Special mode 1 differs from a normal CALL 91 because the value in C1 is negated.

*Negative value indicates special CALL 91 mode

1.11.2 CALL 91 - SPECIAL MODE 2

Allows polyfile master volume and nonzero volume headers to be updated after a logical unit number is changed during an INSTALL procedure.

The syntax of a CALL 91 special mode 2 statement is as follows:

```
    CALL 91,C0,-C1,V,S,P
```

where

```
   C0 - Channel on which the polyfile to be reassigned is open
  -C1 - Channel on which the master volume (volume 0) is open in
        read-only mode*   (e.g., OPEN #C1, "PFNAME@00!")
   V  - Number of the volume to be reassigned
   S  - Status to be returned (S=0 if no error 38)
   P  - Array in which file parameters are returned (see Table
        1-1)
```

A special mode 2 statement causes volume 0 to remain open-locked until the channel is closed.

To reassign volume 0, it must be open on both channels (i.e., channels C0 and C1).

To preserve polyfile integrity, CALL 91 will cause an error 38 and return S=22 if the polyfile is open by another user.

Special mode 2 differs from special mode 1 in that the file opened on channel C0 is not newly built (i.e., the build bit is not set).

*Negative value indicates special CALL 91 mode

1.11.3 CALL 91 - SPECIAL MODE 3

Allows writing to an individual volume of a polyfile that is open in read-only mode. The individual volume is opened on channel C0; no file may be open on channel C1. This is accomplished by entering a pseudo-value (e.g., 777) into C1.

The syntax of a CALL 91 special mode 3 statement is as follows:

```
    CALL 91,C0,C1,V,S,P
```

where

C0	- Channel on which the volume is open in read-only mode
C1	- Channel on which no file is open (may = 777)
V	- Number of the volume to be written to*
S	- 777 on entry (overrides read-only mode); status of call on exit
P	- Array in which file parameters are returned unchanged (see Table 1-1)

*Must be the same volume number as the volume open on C0.

SECTION 2

Section two of this preliminary Polyfile documentation is a general introduction to the concept of the new Polyfile capability. It includes an introduction, comparison with Indexed Files, information on Polyfile structure, and a summary of Polyfile features.

2.1 INTRODUCTION

The newest addition to the family of IRIS files is the Polyfile capability. It is POINT 4's answer to the needs of the more sophisticated applications which require large data bases.

Among the advantages of the Polyfile is the removal of the previously existing size restrictions while refining the keyword access of the Indexed files and keeping their intrinsic sort functions intact. These and other improvements are best illustrated by the comparisons shown in Figure 2-1.

Indexed Files	Polyfiles
Maximum key size is 30 bytes.	Maximum key size is 121 bytes.
Key sizes must be specified in word increments (multiples of 2 bytes).	Key sizes may be specified in exact number of bytes, eliminating problems created by garbage in extra byte.
Maximum number of maximum size (30-byte) keys per file is 1830.	Maximum number of keys per polyfile depends on the key size: 121-byte keys = 8 million 60-byte keys = 20 million 32-byte keys = 40 million
Maximum number of data records is 65,535.	Maximum number of data records is 4,128,768.
Uses the 512-word ABA buffer.	Does not use ABA.
Cannot take full advantage of the R8 buffer pool.	Takes full advantage of the R8 buffer pool.
Has operational Mode 7 for key distribution which necessitates the suspension of time-sharing.	Mode 7 is never needed because keys are redistributed automatically.
Directories cannot be changed in size.	New directory extension volumes may be added dynamically.
The Free List could be clobbered by improper programming.	Free space is allocated by a bit map which cannot be clobbered by user programming.
Data records start at a variable number depending on the size of the directory.	Data records always start at 0.

Figure 2-1. Differences Between Polyfiles and Indexed Files

2.2 POLYFILE STRUCTURE

A polyfile may be composed of a maximum of 64 volumes, numbered 0-63; Volume 0 is always the Master Volume. Polyfile volumes are allocated by the IRIS Operating System as Contiguous Files and are subject to some of the restrictions (i.e. an individual volume cannot exceed 65535 records or the size of the Logical Unit) imposed on Contiguous files in general.

A Polyfile is file type 32.

Volumes of a polyfile may reside on different LUs. Each of these LUs must have enough contiguous space allotted to the user to accommodate a Polyfile volume. It is recommended that the volumes of a Polyfile reside on one physical drive so that backups may be performed frequently and without problems.

A polyfile volume can be one of four legal types:

1. Unstructured
2. Base Directory
3. Directory Extension
4. Data

Unstructured	- An unstructured volume has been linked into the polyfile via the polycall (CALL 91) but has not been structured as a data, base directory, or directory extension volume. For more information on linking and structuring see Section 1.3.
Data Volumes	- A data volume holds records. The same record size must be used for all the volumes in the entire polyfile. The size of a data volume is limited by two factors: number of records (65535) and available contiguous space on an LU. Polyfiles do not have free record chains as do indexed files. Instead, polyfiles have the option of having record maps. All data volumes in a polyfile are either mapped or unmapped. Records are numbered sequentially through ascending data volumes. For example, if a polyfile has three data volumes numbered 2, 6 and 7 which contain 100, 200 and 300 records respectively, then records 0-99 will be found in volume 2, 100-299 in volume 6, and 300-599 in volume 7.

Base Directory	- A base directory volume contains the master and first (leftmost) fine block for one or more directories (up to 63). The numbers of these directories must be increased sequentially by one. Multiple base directory volumes may coexist in a polyfile if their directories are mutually exclusive.
Directory Extension	- A directory extension volume is a space extension of the base directory volume. When a new block is needed in a directory, allocation is first attempted from the base directory volume and then from any extension volumes in ascending order.

2.2.1 POLYFILE HEADER BLOCKS

The header block of the Master Volume (volume 0) contains information about all the other volumes in a polyfile.

Polyfile volume linkage information is stored in polyfile header block locations 200-377. These locations are always 0 in normal contiguous files. In a polyfile volume, word 200 of the header block is, by definition, nonzero. The linkage information is stored in word-pairs as follows:

```
Word 0 (VLU):  Volume Logical Unit (if this word is 0 then
               volume does not exist)

               Bits 15-13: Volume type:
                           0 = Unstructured volume
                           2 = Base directory volume
                           3 = Extension directory volume
                           4 = Data volume
               Bits 12-8:  Number of blocks in the bit map
               Bit     7:  Always one for a volume that exists
               Bits  6-0:  Logical Unit of this volume


Word 1 (BNR):  Base Number of Records - depends on volume type

               Data volume: Number of data records

               Base directory volume:
                 Bits 15-8: Base directory number for this volume
                 Bits  7-0: Number of directories in this volume
                            (maximum number of directories = 63)

               Extension directory volume:
                 Bits 15-8: Base directory number for this volume
                 Bits  7-0: Volume # that this volume extends
```

Word-pairs in the Master Volume (0) header block locations 200-201 describe volume 0; 202-203, volume 1; 204-205, volume 2; . . . ; 376-377, volume 63. This information is used to open the polyfile (see Section 1.4).

There are only two word-pairs in volumes other than 0. The first word-pair (200-201) describes the master volume. The second describes the volume itself (202-203).

In addition to the use of words 200-377 in the header block of a polyfile volume, words 74-172 (FMAP+4 through STAD, see Appendix A) are used to store the key lengths of the different directories in a base directory volume during directory setup. QUERYPF (see Section 1.5) reads these words when it displays polyfile directory information.

2.2.2 DATA FILE TABLE (DFT)

When any file is opened under IRIS a Data File Table (DFT) is employed to retain pertinent information about the file. Since a DFT is normally 8 words in size and a polyfile has a potential size of 64, the DFT is extended by chaining system nodes to it. The DFT Channel Node Pointer (CNP) is used for this purpose. A word-triple is kept for each polyfile volume. System nodes are 32 (decimal) words long. This means that 10 triples may be put in a node. One of the remaining two words is the link to the next node.

The layout for a DFT node word-triple is:

```
Word 0 (VLU):  (If this word is 0 then volume does not exist)

               Bits 15-13: Volume type:
                           0 = Unstructured volume
                           2 = Base directory volume
                           3 = Extension directory volume
                           4 = Data volume
               Bits 12-8:  Number of blocks in the bit map
               Bit     7:  Always one for a volume that exists
               Bits  6-0:  Logical Unit of this volume

Word 1 (BNR):  Depends on volume type

               Data volume: Number of data records

               Base directory volume:
                 Bits 15-8: Base directory number for this volume
                 Bits  7-0: Number of directories in this volume

               Extension directory volume:
                 Bits 15-8: Base directory number for this volume
                 Bits  7-0: Volume # that this volume extends

Word 2 (HRDA): Header block RDA of this volume
```

NOTE

VLU and BNR are copied directly from the header of the master volume.

The word-triples are ordered by volume. The first triple is for volume 0, the second for volume 1, the third for volume 2, etc.

The DFT extension nodes are allocated when a Polyfile is OPENed. Two or more DFTs will share the same extension nodes for the same polyfile. Additional nodes are not allocated when two or more users have accessed the Polyfile concurrently.

An OPEN command will allocate sufficient nodes to provide word triples for volumes 0 through the highest numbered volume in the polyfile. This means that the lower the highest numbered volume in a polyfile is, the fewer number of nodes are used when it is opened.

Other cells in the DFT whose usage will change when a polyfile is opened are described in the following sections.

2.2.2.1 Item Control Blocks

The discsubs for reading and writing polyfile records, READP and WRITP, require an ICB with the following format:

Word	Bits	Description
0	15-0	Lower 16 bits of the record number
1	15-0	Item number or byte displacement
2	15-8	Upper 6 bits of the record number
	7-0	Item type (5=decimal, 11=string, 12=binary)
3	15-0	Desired length (# of words or bytes if type is string)
4	15-0	Address of Source/Destination (relative byte address if string)

2.2.2.2 Bit Maps

Polyfiles use bit maps to allocate records in data volumes and disc blocks in directory volumes.

- All directory volumes must contain maps
- For Data Volumes in a Polyfile, maps are optional (i.e., either all data volumes in a polyfile have maps or none of them do).

The polyfile allocation routine will allocate the first available record or block as specified in the bit map. This differs from indexed files in which the most recently released record is the one allocated.

A polyfile map may range in size from 1 to 16 blocks. The size is given in bits 12-8 of the VLU word. Each bit maps one block or record. A zero bit flags an available record. The first word maps the map blocks, and any additional bits in this word begin mapping available blocks or records (see Figure 2-2).

The map block bits indicate which blocks in the map have free space.

> Zero indicates that the corresponding block has at least one free bit.
>
> One indicates that the corresponding block is "full".

For example, in a data volume with a map size of 5 and records 0, 1, 15 and 17 allocated:

```
                15                                0

Word One = |0|0|0|0|0|0|0|0|0|1|1|0|0|0|0|0|

Word Two = |0|0|0|0|0|0|0|0|0|1|0|1|0|0|0|0|
```

Additional map blocks, after the first block, map only free space, one bit per record or block. Thus any map block can map 4096 units where a unit may be a disc block for a directory volume or a data record for a data volume. The first 1 to 16 bits in the first map block point to the units listed within it.

Up to 4095 bits in the last bit map block may reference nonexistent records or blocks and are thus always marked as in use.

```
+--------------------------------------------------------+
| Free space map     | Map Block bits                    |
|                    +-----------------------------------+
|                                                        |
|                            .                           |
|                                                        |
|                            .                           |
|                                                        |
|                            .                           |
|                                                        |
+--------------------------------------------------------+
```

Figure 2-2. Bit Map Layout

2.2.3 BASE DIRECTORY VOLUMES

A base directory volume always contains the master block and the first fine block of each of its directories. See Figure 1-3 for a layout of a Base Directory Volume.

The BNR word describing the volume in the master volume gives the base directory number (B) and the number of directories (N). The VLU word gives the number of blocks in the bit map (M). With this information it is possible to compute the block in the file containing the master block and first fine block for any directory in the volume.

Given directory D in the volume, the master block for the directory = HRDA+M+(D-B)*2 and the first fine block = HDRA+M+(D-B)*2+1

```
+-------------------------------------------------------+
| Volume Header Block                                   |
+-------------------------------------------------------+
|                                                       |
| Bit Map (up to 16 blocks)                             |
|                                                       |
+-------------------------------------------------------+
| Master level Directory B                              |
+-------------------------------------------------------+
| First fine level block, Directory B                   |
+-------------------------------------------------------+
| Master level Directory B+1                            |
+-------------------------------------------------------+
| First fine level block, Directory B+1                 |
+-------------------------------------------------------+
|                          .                            |
<                          .                            <
>                          .                            >
|                                                       |
+-------------------------------------------------------+
| Master level Directory B+N-1                          |
+-------------------------------------------------------+
| First fine level block, Directory B+N-1               |
+-------------------------------------------------------+
|                                                       |
| Pool of disc blocks for use by any fine or            |
| intermediate level in Directories B through B+N-1     |
|                                                       |
<                          .                            <
>                          .                            >
<                          .                            <
|                                                       |
+-------------------------------------------------------+
```

Figure 2-3. Layout Of A Base Directory Volume

2.2.4 DIRECTORY STRUCTURE

The structure of a directory is a B-tree with bidirectional links across each level. The fine level is always level 0. Levels are numbered sequentially, incrementing by one from the fine level which is level 0. This means that the master block's level number is dependent on the number of intermediate levels. The maximum permissible level number is 127 (see Figure 2-4).

There must always be one master and at least one fine block for each directory.

All links in a polyfile directory consist of a volume number and a volume relative block number.

All blocks in a polyfile directory, with the exception of the master block and first fine block, may be in the base directory volume or any of its extension volumes. The layout of a Directory Block is shown in Figure 2-5.

Key entries on the fine level contain a record number. Other levels contain a link to a block on the next lower level. A key found in a master or intermediate block is the last key found in the block linked on the next lower level.

A terminator key is always present as the last key on all levels. The terminator key consists of all rubouts; i.e., the maximum value that is possible, which is illegal for a key used in a SEARCH command.

```
                             +-----+-----+
                 Level 2     |  0  |  0  |
                             +-----+-----+
                             |K|T|       |
                 Master      |e|E|       |
                             |y|R|       |
                             |1|M|       |
                             |1|*|       |
                             +-----------+
                               / \
                              /   \
                    _________/     \__________________
                     |                              |
                     V                              V
               +-----+-----+                  +-----+-----+
Level 1        |  0  |  <========================>  |  0  |
               +-----+-----+                  +-----+-----+
               |K|K|       |                  |K|T|       |
Intermediate   |e|e|       |                  |e|E|       |
               |y|y|       |                  |y|R|       |
               |0|1|       |                  |1|M|       |
               |5|1|       |                  |8|*|       |
               +-----------+                  +-----------+
                 / \                             / \
                /   \                           /   \
               /     \____                     /     \____
              /           \                   /           \
             /             \                 /             \
            /               \               /               \
            |               |               |               |
Level 0     V               V               V               V
      +-----+-----+   +-----+-----+   +-----+-----+   +-----+-----+
      |  0  |  <=========>  |  <=========>  |  <=========>  |  0  |
      +-----+-----+   +-----+-----+   +-----+-----+   +-----+-----+
      |K|K|K|K|K|K|   |K|K|K|K|K|K|   |K|K|K|K|K|K|   |K|K|K|K|K|T|
Fine  |e|e|e|e|e|e|   |e|e|e|e|e|e|   |e|e|e|e|e|e|   |e|e|e|e|e|E|
      |y|y|y|y|y|y|   |y|y|y|y|y|y|   |y|y|y|y|y|y|   |y|y|y|y|y|R|
      |0|0|0|0|0|0|   |0|0|0|0|1|1|   |1|1|1|1|1|1|   |1|2|2|2|2|M|
      |0|1|2|3|4|5|   |6|7|8|9|0|1|   |2|3|4|5|6|8|   |9|0|1|2|3|*|
      +-----------+   +-----------+   +-----------+   +-----------+
```

*NOTE: TERM is a terminator key which has a value greater than the maximum legal key.

Figure 2-4. Directory Structure

```
        +-+-----------+-----------+
(FNL)   |F|     N     |     L     |  F=Fine level flag. Zero
        +-+-----------+-----------+    for master and intermediate
(MDDFV) |M|     D     |   DFvv    |    levels of directory,
        +-+-----------+-----------+    one for fine level.
(PFDF)  |            DF           |  N=# of active keys in block.
        +-----------+-------------+  L=Key length in bytes. Maximum
(PIDPV) |     I     |   DPvv      |    size is 121 bytes.
        +-----------+-------------+  M=Master level flag.  One
(PFDP)  |            DP           |    for master level, zero for
        +-------------------------+    intermediate and fine levels.
(PFKEY) |            P1           |  D=Directory Number.
        +-----------+-------------+  DFvv=Volume number of next
        |   P1vv    |             |    block in this level of this
        +-----------+             |    directory.
        |                         |  DF=Relative disc address of
        |                         |    next block of this level of
        |            K1           |    this directory.  (Zero in
        |                         |    last block in level.)
        |                         |  I=Intermediate level number.
        +-------------------------+    Fine level is zero, master
        |            P2           |    level is the highest number.
        +-----------+-------------+  DPvv=Volume number of previous
        |   P2vv    |             |    block in this level of this
        +-----------+             |    directory.
        |                         |  DP=Relative disc address of
        |                         |    previous block of this level
        |            K2           |    of this directory. (Zero in
        |                         |    first block in level.)
        |                         |  K=Key value (ASCII string,
        +-------------------------+    odd number of bytes).
        |                         |  Pvv=Pointer overflow. Volume
        |            .            |    number of block at next
        |                         |    level in master and
        |            .            |    intermediate levels, or upper
        |                         |    six bits of real record
        +-------------------------+    number in fine level.
        |            Pn           |  P=Pointer. Relative disc
        +-----------+-------------+    address of block at next
        |   Pnvv    |             |    level in master and
        +-----------+             |    intermediate levels, or lower
        |                         |    16 bits of real record number
        |                         |    in fine level.
        |            Kn           |
        |                         |
        |                         |
        +-------------------------+  There may be as many as
        |                         |  INT (251/L1) entries
        |                         |  in a block.  Where L1 =
        |                         |  INT((L+4)/2).  Even when
        |          UNUSED         |  full there may be up to L1-1
        |                         |  words unused at the end of
        |                         |  each directory block.
        +-------------------------+
```

Figure 2-5. Directory Block Layout

2.3 DATA VOLUME ACCESS BY RECORD

Each Data Volume in a Polyfile has the same record size as all the other volumes in the Polyfile.

Each Data Volume contains an integral number of records which is recorded in the BNR word (see Section 2.2.1).

Maps are optional but if one data volume is mapped then all data volumes in a Polyfile must be mapped. If a map is present, then data records start at the beginning of the first block after the map block(s). The maximum number of map blocks is 16.

As in the IRIS Contiguous File, a record may exceed one block and cross block boundaries.

Records in a Polyfile are considered to be logically contiguous through all Data Volumes in ascending order.

To access record 'R', two steps are required:

a. Finding the correct Data Volume
b. Finding the correct record within the volume

To find the correct volume, the base record number of each volume is computed in sequence by summing the Base Number of Records (BNR) of all preceding Data Volumes. Therefore, new Data Volumes cannot be added which have a volume number less than the highest pre-existing data volume. A good rule of thumb is to number all data volumes as low as possible.

2.4 DIRECTORY VOLUME ACCESS BY KEY

If a record is to be accessed by key, the Directory volume is searched as follows:

a. Read master block
b. Scan block for first key >= target
c. If fine block then key is found
d. Get block from next level and go to step b

Search mode 2 (match) uses the above routine and returns the record number associated with the key on the fine level.

Search mode 3 (next) uses the same routine but the test in step b is changed to >.

Search mode 4 (insert) uses the same routine with modifications:

- If any block is full (except a fine block) then the block is split

- If the block to be split is the master block, a new (higher) level master block is created. The master block must be in a specific block. To maintain this condition on a master level split, two new blocks are allocated, and the contents of the master block split between them.

- The master block is reset to two keys, the last key in the first of the new blocks and the terminator key (last in the second block).

The purpose of the modifications is to assure that in every block in the insertion path there exists the space for at least one more key. If the key space were not assured, a key insertion that fails for lack of space would corrupt the directory structure.

Search mode 5 (delete) also uses the same routine with a further set of modifications. Two factors, alpha and beta, are used to control the redistribution of keys between blocks on a level. The redistribution is accomplished in the following sequence:

- Non-Masterblocks are scanned
- Keys are redistributed
- Adjoining blocks are combined where necessary
- Intermediate levels are eliminated if necessary

2.5 SUMMARY

1. Polyfiles may have up to 64 (0-63) volumes in one Polyfile.

   - Maximum size of a Polyfile can exceed 4 million records
   - Each volume may reside on a different Logical Unit (IRIS R8 permits up to 127 Logical Units)
   - A Polyfile Master Volume is always Volume 0
   - All Logical Units containing volumes of a Polyfile must be installed, otherwise the Polyfile cannot be opened
   - A Polyfile may not be built on LU/0

2. The file type for a Polyfile is 32

3. The maximum number of directories is 63

4. Data volumes in one Polyfile are either mapped or unmapped.

   - Maximum number of records in an unmapped Data Volume may be 65535 depending on record size
   - Maximum number of records in a mapped Data Volume may be 60000 to 65000 depending on:

     a. Size of Logical Unit
     b. Number of map blocks required (maximum is 16)
     c. Record Size

   - Data Volumes are added in ascending volume numbers. Data Volumes should be given low numbers. After Volume 63 has been built no further data volumes may be added even if there is a gap in the lower sequence.

5. Maximum key size is 121 bytes with a potential for over 12 million keys per Polyfile

   - Keylengths are returned by SEARCH mode 1
   - SEARCH mode 1 will return a V2 of 13 (directory not found) rather than V2 of 5 (structure error)

6. Highest numbered volume of a Polyfile determines the number of system nodes allocated when a Polyfile is open. By keeping the volume numbers low, the system overhead is reduced.

7. Polyfile integrity is checked by the HSLA (Hours Since Last Access) value. This value is 32 bits in size and has a resolution of .1 second. If the HSLA of the volumes does not match, the Polyfile will not open.

8. Polyfiles take full advantage of the buffer pool.

9. New Directory Extension Volumes may be added dynamically.

10. Key capacity has been improved by expanding the number of possible levels (0-26) between the master and fine levels.

11. It is recommended that:

    - Polyfile names be made to differ from each other and from other file types

    - Polyfiles should be backed up frequently at regular intervals to avoid the loss of data

    - As many volumes of any one Polyfile as possible should reside on the same physical unit

POLYFILE EXERCISE

The following exercise will acquaint the first-time user with the process of building a polyfile. Follow the steps given in their sequence and note the system responses. Each user response requires a <RETURN> for it to be entered. Where a <RETURN> is shown as the response it needs to be pressed just that one time, otherwise the press <RETURN> is not shown. At the # sign enter:

Step	Command	System Prompt (and comments in parentheses)
1	BUILDPF	BUILDPF - Build Polyfile Utilities Polyfilename [must have "LU" (not 0)]:
2	1/DEMOFILE@	Polyfile not found. Do you wish to create one?
3	Y	Creating NEW polyfile Record size (in words) for entire Polyfile:
4	20	Volume: 0 Volume types: "B" Base Directory "E" Extension Directory "D" Data Volume type:
5	B	(We have decided to build the Base Directory) Starting directory number for this volume:
6	7	(We have started with directory #7) Directory numbers available from 7 thru 63 Directory 7 Key size in characters [<RETURN> to terminate]:
7	10	(The key size for vol. 7 is to be 10 characters long) Directory 8 Key size in characters [<RETURN> to terminate]:
8	31	(The key size for vol. 8 is to be 31 characters) Directory 9 Key size in characters [<RETURN> to terminate]: (As can be seen the system will call out the next volume automatically until the user terminates this process. For the purpose of this exercize we will continue until we get to directory 10)

Step	Command	System Prompt (and comments in parentheses)
9	25	(Key size for this directory is set at 25)
		Directory 10 Key size in characters [<RETURN> to terminate]:
		(At this point we decide that we have enough directories for this volume)
10	<RETURN>	This directory setup ok? [<RETURN> = ok]:
		(Either a 'Y' or a <RETURN> may be used. (We will enter a 'Y'.)
11	Y	Volume size in indexes (keys) [Negative for size in blocks]:
12	250	(The size has been entered in 'keys', therefore a positive number was used)
		Allocating volume. Please wait. Volume 0 allocation complete. Structuring volume 0 as Base Directory Volume. Please wait. Directory: 7 8 9 Structuring complete.
		Extend Base Volume 0 more ["Y""N"; <RETURN> = exit]: (We decide to extend)
13	Y	Logical Unit (non-zero) for volume [<RETURN> = exit]:
14	1	(This volume is to be extended on LU1)
		Volume number [0-63; <RETURN> = don't care]: Base Volume 0 and its current extensions have a total of 41 blocks which will hold a maximum of approximately 246 keys.
15	<RETURN>	(The following prompt requests the number of keys or blocks for the new Extension Directory. That number must be added to the number of keysblocks given in step 12, i.e. the number given there was 250, we decide we want 100 more and the number to be entered is therefore 350).
		New maximum number of indexes (keyes) [Negative for size in blocks]:

```
Step   Command      System Prompt (and comments in parentheses)

 16     350         Allocating volume.  Please wait.
                    Volume 1 allocation complete.
                    Structuring volume 1 as a directory extension.
                    Please wait.
                    Structuring complete.
                    Extend Base Volume 0 more ["Y""N"; <RETURN> =
                    exit]:

 17      N          (We do not wish to extend anymore)

                    Logical Unit (nonzero) for volume [<RETURN> =
                    exit]:

 18      1          (This volume is to reside on LU1)

                    Volume number [ 0-63; <RETURN> = don't care]:

 19   <RETURN>      (We don't care about the vol. number and let
                    the system assign the next number)

                    Volume types: "B"  Base Directory
                                  "E"  Extension Directory
                                  "D"  Data
                    Volume type:

 20      D          (We wish to build a Data Volume)

                    Data Volume(s) to have maps? ["Y""N"]:

 21      Y          (Yes, the Data Volumes are to have maps)

                    Volume size in indexes (keys)
                       [Negative for size in blocks]:

 22     500         (The size has been entered in 'keys',
                    therefore a positive number was used)

                    Allocating volume.  Please wait.
                    Volume 2 allocation complete.
                    Structuring volume 2 as Data Volume.  Please
                    wait.
                    Structuring complete.
                    Logical Unit (non-zero) for volume [<RETURN> =
                    exit]:

 23     1           (This volume is to be extended on LU1)

                    Volume number [ 0-63; <RETURN> = don't care]:
```

Step	Command	System Prompt (and comments in parentheses)
24	<RETURN>	(We will let the computer assign volume numbers) Volume types: "B" Base Directory "E" Extension Directory "D" Data Volume type:
25	B	(Starting with a Base Directory again) Starting directory number for this volume:
26	1	(Starting with directory 1 this time) Directory numbers available from 1 thru 63 Directory 1 Key size in characters [<RETURN> to terminate]:
27	17	(The key size for directory 1 is to be 17 characters long) Directory 2 Key size in characters [<RETURN> to terminate]:
28	12	(The key size for directory 2 is to be 12 characters) Directory 3 Key size in characters [<RETURN> to terminate]:
29	7	(The key size for directory 3 is to be 7 characters long) Directory 4 Key size in characters [<RETURN> to terminate]:
30	5	(The key size for directory 4 is to be 5 characters) Directory 5 Key size in characters [<RETURN> to terminate]:
31	<RETURN>	(This time we terminate here) This directory setup ok? [<RETURN> = ok]: (Either a 'Y' or a <RETURN> may be used. We will enter a 'Y'.)
32	Y	Volume size in indexes (keys) [Negative for size in blocks]:

```
Step  Command      System Prompt (and comments in parentheses)

 33   1000         (The size has been entered in 'keys',
                   therefore a positive number was used)

                   Allocating volume.  Please wait.
                   Volume 3 allocation complete.
                   Structuring volume 3 as Base Directory Volume.
                   Please wait.
                   Directory:   1   2   3   4   Structuring
                   complete.

                   Extend Base Volume 3 more ["Y""N"; <RETURN> =
                   exit]:

                   (We decide not to extend)

 34      N         Logical Unit (non-zero) for volume [<RETURN> =
                   exit]:

 35      2         (This volume is to be extended on LU2)

                   Volume number [ 0-63; <RETURN> = don't care]:

 36   <RETURN>     Volume types: "B"  Base Directory
                                 "E"  Extension Directory
                                 "D"  Data
                   Volume type:

 37     B          (We have decided to build yet another Base
                   Directory)
                   Starting directory number for this volume:

 38    44          (We have started with directory #44)

                   Directory numbers available from 44 thru 63

                   Directory  44   Key size in characters
                   [<RETURN> to terminate]:

 39    60          (The key size for directory #44 is to be 60
                   characters long)
                   Directory  45   Key size in characters
                   [<RETURN> to terminate]:

 40   <RETURN>     (The size of '60' was an error, so we cancel
                   this and start over

                   This Directory setup ok? [<RETURN> = ok]:
                   terminate]:

 41    N           Starting Directory number for this volume:
```

Step	Command	System Prompt (and comments in parentheses)
42	44	Directory numbers available from 44 thru 63 Directory 44 Key size in characters [<RETURN> to terminate]:
43	120	(The key size for vol. 44 is to be 120 characters long) Directory 45 Key size in characters [<RETURN> to terminate]:
44	<RETURN>	This directory setup ok? [<RETURN> = ok]
45	<RETURN>	Volume size in indexes (keys) [Negative for size in blocks]:
46	100	Allocating volume. Please wait. Volume 4 allocation complete. Structuring volume 4 as Base Directory Volume. Please wait. Directory: 44 Structuring complete. Logical Unit (nonzero) for volume [<RETURN> = exit]
47	1	Volume number [0-63; <RETURN> = don't care]:
48	4	Volume types: "B" Base Directory "E" Extension Directory "D" Data Volume type:
49	E	(We have chosen to build an Extension Directory) Volume to extend:
50	0	(The Extension Directory is to be #0) Base Volume 0 and its current extensions have a total of 58 blocks which will hold a maximum of approximately 348 keys. New maximum number of indexes(keys) [Negative for size in blocks]:
51	-25	(This time the size is being given in blocks) Allocating volume. Please wait. Volume 5 allocation complete. Structuring volume 5 as Base Directory Volume. Please wait. Structuring complete. Extend Base Volume 0 more ["Y""N"; <RETURN> = exit]:

Step	Command	System Prompt (and comments in parentheses)
52	N	Logical Unit (non-zero) for volume [<RETURN> = exit]:
53	1	Volume number [0-63; <RETURN> = don't care]:
54	9	Volume types: "B" Base Directory "E" Extension Direcory "D" Data
55	D	Volume size in records [Negative for size in blocks]:
55	-10	Allocating volume. Please wait. Volume 9 allocation complete. Structuring volume 9 as a data volume. Please wait. Structuring complete. Logical Unit (nonzero) for volume [<RETURN> = exit]:
57	<RETURN>	(Exit from this exercise)

Now do a QUERYPF for a complete dump and the result should be the same as shown in Section 1.5 if both LU1 and LU2 were installed. If LU2 was not installed then you will get meaningless information for volume 4.

Finally, delete the Polyfile volumes created for this exercise by using KILLPF as shown in Section 1.7.

COMMENT SHEET

MANUAL TITLE Preliminary Polyfiles Document

PUBLICATION NO. - - REVISION 07

FROM: NAME/COMPANY:

BUSINESS ADDRESS:

CITY/STATE/ZIP:

COMMENTS: Your evaluation of this manual will be appreciated by POINT 4 Data Corporation. Notation of any errors, suggested additions or deletions, or general comments may be made below. Please include page number references where appropriate.

NO POSTAGE NECESSARY IF MAILED IN U.S.A.
Fold on Dotted Lines and Tape

NO POSTAGE
NECESSARY
IF MAILED IN
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 5755 SANTA ANA, CA.

POSTAGE WILL BE PAID BY ADDRESSEE:

POINT 4 Data Corporation
PUBLICATIONS DEPARTMENT
2569 McCabe Way
Irvine, CA 92714

CUT ON THIS LINE

POINT 4 DATA CORPORATION

2569 McCabe Way / Irvine, California 92714 / (714) 863-1111