System i

Programming
Optical device programming

Version 6 Release 1

System i

Programming
Optical device programming

Version 6 Release 1

Note

Before using this information and the product it supports, be sure to read the information in "Notices," on page 49.

This edition applies to version 6, release 1, modification 0 of IBM i5/OS (product number 5761-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

Contents

Optical device programming

Certain application programming interfaces (APIs) work with i5/OS® optical file systems. i5/OS optical file systems consist of any data storage system that uses optical media, which includes CD-ROM, digital video disc (DVD), Write Once Read Many (WORM) media, and magneto-optical media.

Programmers can use these APIs to create, access, change, or maintain optical files and directories. You can also use these APIs to customize the use of optical support for specific business applications.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 48.

Related concepts

Application programming interfaces

PDF file for Optical device programming

You can view and print a PDF file of this information.

To view or download the PDF version of this document, select Optical device programming (about 538 KB).

Saving PDF files

To save a PDF on your workstation for viewing or printing:

1. Right-click the PDF link in your browser.
2. Click the option that saves the PDF locally.
3. Navigate to the directory in which you want to save the PDF.
4. Click **Save**.

Downloading Adobe Reader

You need Adobe® Reader installed on your system to view or print these PDFs. You can download a free copy from the Adobe Web site (www.adobe.com/products/acrobat/readstep.html)  .

Related reference

"Related information for Optical device programming" on page 47

Web sites and other information center topic collections contain information that relates to the Optical device programming topic collection. You can view or print any of the PDF files.

Optical device programming concepts

You can manipulate optical files and directories by using either integrated file system support or hierarchical file system (HFS) support. Programming for optical devices can be easier if you understand volume, directory, and file considerations.

Two categories of APIs can be used to manipulate optical files and directories:

- Integrated file system support, which consists of UNIX-type APIs and the generic command interface
- Hierarchical file system (HFS) support, which consists of APIs and generic commands.

You can use both categories of APIs concurrently. For example, an optical file that is opened for reading by one application by using the HFS Open Stream File API can be opened for reading by another application using the Open UNIX-type API.

Because different file systems exist in i5/OS, you must provide some means for the HFS or the integrated file system to differentiate for which file system a call is targeted. This is accomplished by requiring that the first name in the path name parameter be the name of the file system to be called, preceded by a leading slash. In order for the optical file system to be identified as the receiver of a request submitted to the HFS or the integrated file system, the first portion of the path name parameter must be **/QOPT**. The remaining path elements to follow /QOPT are volume/directory/subdirectory/file. See the following example for a path name:

/QOPT/CD001/Dir1/SubDir1/File

Related concepts

Application programming interfaces

Integrated file system and optical device programming

The integrated file system is a part of the i5/OS operating system that supports stream input/output and storage management similar to personal computer and UNIX® operating systems, while providing an integrated structure over all information stored on your system.

The integrated file system comprises 11 file systems, each with its own set of logical structures and rules for interacting with information in storage. Key features of the integrated file system include the following items:

- Support for storing information in stream files that can contain long continuous strings of data. These strings of data might be, for example, the text of a document or the picture elements in a picture. The stream file support is designed for efficient use in client/server applications.
- A hierarchical directory structure that allows objects to be organized similar to common PC file structures. A path specified through the directories to an object accesses the object.
- A common interface that allows users and applications to access not only the stream files but also database files, documents, and other objects that are stored on your system.
- A common view of stream files that are stored locally on your system, on an integrated System x™ or IBM® BladeCenter® server, or on a remote Windows® server. Stream files can also be stored remotely on a local area network (LAN) server, a Novell NetWare server, another remote system running the i5/OS operating system, or a Network File System server.

The integrated file system enhances the already extensive data management capabilities of i5/OS with additional capabilities to better support emerging and future forms of information processing, such as client/server, open systems, and multimedia.

The integrated file system enables you to do the following tasks:

- Attain fast access to i5/OS data, especially for applications such as IBM System i® Access that use the i5/OS file server.
- Handle types of stream data, such as images, audio, and video more efficiently.
- Use a file system base and a directory base for supporting UNIX-based open system standards, such as Portable Operating System Interface for Computer Environments (POSIX) and XPG. This file structure and this directory structure also provides a familiar environment for users of PC operating systems such as Disk Operating System (DOS), and Windows operating systems.
- Gain access to file support with unique capabilities (such as record-oriented database files, UNIX-based stream files, and file serving) to be handled as separate file systems, while allowing them all to be managed through a common interface.

LAN-attached optical devices do not support this interface.

Related concepts

Integrated file system

Hierarchical file system and optical device programming

A hierarchical file system (HFS) is a part of the operating system that includes the application programming interface (API) and the underlying file system (optical or otherwise) support.

The HFS API makes it possible for an application that is written in a high-level language to create, store, retrieve, and manipulate data on a directly attached optical library device, LAN-attached optical library device, CD-ROM, or DVD device.

HFS API optical support consists of two parts:

- An application programming call interface to the hierarchical file system to manipulate objects known as files and directories.
- An optical or other registered file system that manages the storage devices where the files and directories are stored.

HFS API optical functions include the following items:

- Creating or deleting a directory
- Opening, reading, or closing a directory
- Opening, reading, writing, or closing a file
- Locking or unlocking bytes in a file
- Getting or setting the size of a file
- Renaming, copying, deleting, or removing a file
- Retrieving or changing directory entry attributes

Applications use HFS APIs to manage stream files on an i5/OS system. These stream files are also called objects to identify them as data elements that do not have a conventional record structure. The object is treated as a named byte stream of known length, whose size can vary from a few bytes to megabytes.

HFS APIs allow applications to create and manage file objects on storage devices and to perform input/output operations to those file objects. HFS APIs allow applications to create and manage directory objects, which can be thought of as a logical grouping of similar file objects. These directory objects contain information about the file objects that belong to that directory. Directories can be contained within directories resulting in the hierarchical structure.

Related concepts

Application programming interfaces

Volume, directory, and file considerations

Programming for optical devices can be easier if you understand these considerations for volumes, directories, and files.

Volume considerations

Consider the following terms when referring to volumes:

Online
: The volume is mounted in a drive under the read/write heads.

Near online
: The volume is in the optical media library, but not online. The volume can be in a storage slot or the opposite side of an online volume.

Removed
The volume is not physically in an optical media library, but volume information for the volume is kept when the volume is removed.

Offline
The volume is in an optical device, but the device is powered off, varied off, or no longer connected.

Consider the following characteristics of optical volumes:

- An optical volume is one side of an optical cartridge.
- Some optical cartridges contain two volumes, others contain one.
- All volume names must be unique.
- Depending on the optical media density and type, the capacity of a volume can range from a few hundred megabytes to many gigabytes.
- Normally, a near online volume takes less than 10 seconds to become an online volume. This requires the volume to be mounted into a drive.
- The number of drives in the optical media library determines how many volumes can be online at any time. Only one volume can be mounted in a drive (online) at one time. The remaining volumes in the library are near online.
- Volumes are generally independent of each other, with one exception. The two volumes on the same cartridge can never be completely independent. Both volumes on a cartridge can never be online at the same time. Copying between two volumes on the same cartridge can be done, but it requires the cartridge to be "flipped" several times to copy all of the requested files.
- There is no limit to the number of removed volumes that can exist.

How an application manages volumes depends almost entirely on the requirements of the application. Data need to be written to volumes strategically, depending on the required retrieval time in the future. If it is not desirable to wait for a near online volume to become online, the application might need to be set up so that the most likely volumes to be accessed are online.

Directory considerations

The only limit to the number of directories that can be created on a volume is the capacity of the media. This restriction also applies to the number of files that can exist in an optical directory. Directories are not required to exist for files to be stored on a volume. If you want, all files can be stored in the root directory of a volume. The root directory is the "/" directory that is created when a volume is initialized. This root is not considered a directory in the traditional sense since it cannot be created or deleted like other directories. The root directory will always exist on initialized optical volumes.

Directories can be used to categorize optical files into more manageable subsets. Directories can contain files from a particular time period, subject, characteristic, or any combination of these. For example, there may be a directory SPOOLFILES with subdirectories YEAR_1994 and YEAR_1995. Taking this one step further, there can be subdirectories within these subdirectories named MONTH_MARCH and MONTH_APRIL. See the following example for this structure:

```
/SPOOLFILES /YEAR_1994 /MONTH_MARCH
86 Optical Support V5R3
|
|
/MONTH_APRIL /YEAR_1995 /MONTH_MARCH /MONTH_APRIL
```

The following example contains the fully qualified directory names:

```
/SPOOLFILES
/SPOOLFILES/YEAR_1994
/SPOOLFILES/YEAR_1994/MONTH_MARCH
```

```
/SPOOLFILES/YEAR_1994/MONTH_APRIL
/SPOOLFILES/YEAR_1995
/SPOOLFILES/YEAR_1995/MONTH_MARCH
/SPOOLFILES/YEAR_1995/MONTH_APRIL
```

Directories can be useful when categorizing files, but they are not necessary. Like volume names, directory names must be unique within the same volume. For example, volume VOL001 cannot have two directories named DIR001. Volume VOL001 can, however, have a DIR001 directory and a DIR000/DIR001 directory. Also, a DIR001 directory can exist on volume VOL001 and volume VOL002. For information about directory naming conventions, see "Path names requirements" on page 37.

File considerations

The size of optical files depends almost entirely on the requirements of the application and the users of those files. The size of an optical file (accessible through HFS or the integrated file system) can range from 0 bytes to 4 294 705 152 bytes depending on the capacity of a volume. The physical size of the target piece of media is limited by the amount of free space available.

When selecting optimal file sizes for your application, pay special attention to the following considerations:

- The amount of system disk unit or main storage on the system
- How the data will be read (sequentially or randomly)
- Whether the entire file will typically be retrieved, or just a small portion
- Whether files will be updated once they are written to the volume

Generally, the larger the file, the better the performance and media use. When larger files are used, less media space is taken up by file directory information and more is used for actual data. Also, the performance related to file size is not a linear comparison. It does not take twice as long to write 20 KB of data as it does to write 10 KB of data. Performance (KB/second) improves as the amount of data being read or written increases.

Integrated file system programming for i5/OS

The integrated file system support provides a UNIX-type interface that you can use to maintain optical files and directories. LAN-attached optical devices do not support this interface.

The integrated file system support for optical support consists of UNIX-type APIs and generic commands.

Like all file systems, the optical file system has unique rules and restrictions for applications that access optical functions through the integrated file system. Several of the UNIX-type APIs and generic commands are not supported. Others are only partially supported, or restricted.

Integrated file system APIs

UNIX-type APIs are C language functions that can be used in Integrated Language Environment® (ILE) C for i5/OS programs.

The following table is a quick reference for supported and unsupported UNIX-type optical file system APIs.

Table 1. Optical implementation of UNIX-type APIs

UNIX-type API	Supported	Comments and usage notes

Table 1. Optical implementation of UNIX-type APIs (continued)

access (Determine File Accessibility)	Yes	Requires *X authority to the parent optical volume. For non-Universal Disk Format (UDF) volumes, no other authority is required. For UDF formatted volumes, the following authorization rules apply: • Requires *X authority to each directory in the path preceding the object tested. • Requires *R authority when R_OK is specified. • Requires *W authority when W_OK is specified. • Requires *X authority when X_OK is specified. • Requires *RX authority when R_OK | X_OK is specified. • Requires *WX authority when W_OK | X_OK is specified. • Requires *RX authority when R_OK | W_OK is specified. • Requires no authority when F_OK is specified.
accessx (Determine File accessibility based on the **who** parameter)	Yes	Does not require *X authority to the parent optical volume. For UDF volumes, the following authorization rules apply: Valid values for the **who** parameter are: • ACC_INVOKER • ACC_SELF • ACC_ALL • ACC_OTHERS 1. Requires *R authority when R_OK is specified 2. Requires *W authority when W_OK is specified 3. Requires *X authority when X_OK is specified Authority checks are mutually exclusive.
chdir (Change Current Directory)	Yes	Requires *X authority to the parent optical volume. For non-UDF volumes, no other authority is required. For UDF formatted volumes, *X authority is required to each directory in the path

Table 1. Optical implementation of UNIX-type APIs (continued)

chmod (Change File Authorizations)	Yes	Only supported for UDF formatted optical volumes. Requires *CHANGE authority to the parent optical volume. Requires *X authority to each directory in the path preceding the object. To perform this operation, you must be the owner of the file or have *ALLOBJ special authority.
chown (Change Owner and Group of File)	Yes	Only supported for UDF formatted optical volumes. Requires *CHANGE authority to the parent optical volume. Requires *X authority to each directory in the path preceding the object. To perform this operation, you must be the owner of the file, or have *ALLOBJ special authority. Files and directories on non-UDF formatted volumes are owned by QDFTOWN user profile.
close (Close File Descriptor)	Yes	
closedir (Close Directory)	Yes	
creat (Create or Rewrite File)	Yes	Requires *CHANGE authority to the parent optical volume. For non-UDF volumes, no other authority is required. For UDF formatted volumes, *X authority is required to each directory in the path and *WX authority to the parent directory. The change and modification time stamps for the parent directory are not updated.
dup (Duplicate Open File Descriptor)	Yes	
dup2 (Duplicate Open File Descriptor to Another Descriptor)	Yes	
fchmod (Change File Authorizations by Descriptor)	Yes	Only supported for UDF formatted optical volumes. To perform this operation, you must be the owner of the file or have *ALLOBJ special authority.
fchown (Change Owner and Group of File by Descriptor)	Yes	Only supported for UDF formatted optical volumes. To perform this operation, you must be the owner of the file or have *ALLOBJ special authority. Files and directories on non-UDF formatted volumes are owned by QDFTOWN user profile.
fcntl (Perform File Control Command)	No	

Table 1. Optical implementation of UNIX-type APIs (continued)

fpathconf (Get Configurable Path Name Variables by Descriptor)	Yes	
fstat (Get File Information by Descriptor)	Yes	Owner, group, and other mode bits are always on, regardless of the user's authority to the file. File access time stamp is not changed.
fsync (Synchronize Changes to File)	Yes	For UDF formatted volumes, data is forced to optical disk. For non-UDF formatted volumes, data is forced to internal disk storage that is recoverable through held optical files.
ftruncate (Truncate File)	Yes	
getcwd (Get Current Directory)	Yes	Requires *X authority to the parent optical volume. For non-UDF volumes, no other authority is required. For UDF formatted volumes, *RX authority is required to each directory in the path name preceding the object.
getegid	Yes	
geteuid	Yes	
getgid	Yes	
getgrid	Yes	
getgrnam	Yes	
getgroups	Yes	
getpwnam	Yes	
getpwuid	Yes	
getuid	Yes	
ioctl (Perform File I/O Control Request)	No	
link (Create Link to File)	No	QOPT does not support links.
lseek (Get File Read/Write Offset)	Yes	
lstat (Get File or Link Information)	Yes	File access time stamp is not changed. Requires *X authority to the parent optical volume. For non-UDF volumes, no other authority is required. For UDF formatted volumes, *X authority is required to each directory in the path preceding the object and *R authority is required to the object.

Table 1. Optical implementation of UNIX-type APIs (continued)

mkdir (Make Directory)	Yes	Requires *CHANGE authority to the parent optical volume. For non-UDF volumes, no other authority is required. For UDF formatted volumes, *X authority is required to each directory in the path and *WX authority to the parent directory. The change and modification time stamps for the parent directory are not updated. Owner ID and group ID are not set.
open (Open File)	Yes	If the file is opened for write access , *CHANGE authority is required to the parent optical volume. If the file is opened for read access, *USE authority is required to the parent optical volume. For UDF formatted volumes, the following additional authorization rules apply: • Requires *R authority when object is being opened O_RDONLY. • Requires *W authority when object is being opened O_WRONLY. • Requires *RW authority when object is being opened O_RDWR. • Requires *WX to the parent directory when object does not exist and O_CREAT is specified.
opendir (Open Directory)	Yes	Requires *USE authority to the parent optical volume. For UDF formatted volumes, *X authority is required to each directory in the path preceding the object, and *R authority is required to the object being opened.
pathconf (Get Configuration Path Name Variables)	Yes	
Qp0lGetPathFromFileId	Yes	

Table 1. Optical implementation of UNIX-type APIs (continued)

Qp0lRenameKeep	Partial	QOPT does not support renaming a directory. The object must be a file. Requires *CHANGE authority to the parent optical volume. For non-UDF volumes, no other authority is required. UDF formatted volumes require *X authority to each directory in the path, and *WX authority to the parent directory, and *W authority to the file. If renaming the volume, *RWX is required to the root (/) directory of the volume. New and old files must exist in the same directory.
Qp0lRenameUnLink	Partial	QOPT does not support renaming a directory. The object must be a file. Requires *CHANGE authority to the parent optical volume. For non-UDF volumes, no other authority is required. UDF formatted volumes require *X authority to each directory in the path, *WX authority to the parent directory, and *W authority to the file. If renaming the volume, *RWX is required to the root (/) directory of the volume. The object that is identified by a new path cannot exist.
read (Read from File)	Yes	The file access time is not updated. When reading from files on volumes formatted in Universal Disk Format (UDF), byte locks on the range being read are ignored. The same is true for readv().
readdir (Read Directory Entry)	Yes	The directory access time is not updated.
readlink (Read Value of Symbolic Link)	No	QOPT does not have symbolic links.

Table 1. Optical implementation of UNIX-type APIs (continued)

rename (Rename File or Directory)	Partial	QOPT does not support renaming a directory. The object must be a file or a volume. Requires *CHANGE authority to the parent optical volume. For non-UDF volumes, no other authority is required. UDF formatted volumes require *X authority to each directory in the path, *WX authority to the parent directory, and *W authority to the file. If renaming the volume, *RWX is required to the root (/) directory of the volume. The object that is identified by a new path cannot exist.
rewinddir	Yes	
rmdir (Remove Directory)	Yes	Requires *CHANGE authority to the parent optical volume. For non-UDF volumes, no other authority is required. For UDF formatted volumes, *X authority is required to each directory in the path and *WX authority is required to the parent directory. Change and modification time stamps for the parent directory are not updated. The operation will not be allowed if the directory is busy.
stat (Get File Information)	Yes	File access time stamp is not changed. Requires *X authority to the parent optical volume. For non-UDF volumes, no other authority is required. For UDF formatted volumes, *X authority is required to each directory in the path preceding the object and *R authority is required to the object. When issued to an optical volume, the size returned is the volume capacity or 2 147 483 647, whichever is smaller.
symlink (Make Symbolic Link)	No	QOPT does not support symbolic links.
sysconf (Get System Configuration Variables)	Yes	

Table 1. Optical implementation of UNIX-type APIs (continued)

unlink (Remove Link to File)	Yes	Requires *CHANGE authority to the parent optical volume. For non-UDF volumes, no other authority is required. For UDF formatted volumes, *X authority is required to each directory in the path and *RX authority is required to the parent directory. Change and modification time stamps for parent directory are not updated. Link to a file cannot be removed when a job has the file opened.
unmask (Set Authorization Mask for Job)	Yes	
utime (Set File Access and Modification Times)	No	QOPT does not support setting the file access or modification time.
write (Write to File)	Yes	Change and modification time stamps for the file are updated when the file is closed. When writing to files on volumes formatted in Universal Disk Format (UDF), byte locks on the range being written are ignored. The same is true for writev().

Integrated file system generic commands

Generic commands are system control language (CL) commands that provide an interface to optical support.

Table 2 is a quick reference of generic CL commands that are related to the integrated file system.

For authorities that are required to issue generic commands, see the Security Reference.

Table 2. Optical implementation of generic commands

Generic command	Supported	Comments and restrictions
ADDLNK	No	
CHGAUD	No	
CHGAUT	Yes	Supported only for UDF-formatted optical volumes.[1]
CHGCURDIR	Yes	
CHGOWN	Yes	Supported only for UDF-formatted optical volumes.
CHGPGP	Yes	Supported only for UDF-formatted optical volumes.
CHKIN	No	
CHKOUT	No	
CPY	Yes	

Table 2. Optical implementation of generic commands (continued)

Generic command	Supported	Comments and restrictions
CRTDIR	Yes	Command will fail if attempt is to create /QOPT or next level directory, which represents a volume.
DSPAUT	Yes	
DSPCURDIR	Yes	
DSPLNK	Yes	
ENDJRN	No	
MOV	Partial	QOPT does not support moving a directory, if it contains files or subdirectories. QOPT does not support moving a volume.
RMVDIR	Partial	QOPT does not support RMVLNK(*YES).
RMVLNK	Yes	
RNM	Partial	QOPT does not support renaming a directory.
RST	Partial	QOPT supports restoring an entire volume using SUBTREE (*STG).
RTVCURDIR	Yes	
SAV	Partial	QOPT supports saving an entire volume using SUBTREE (*STG).
SAVRST	No	
STRJRN	No	
WRKAUT	Yes	Supported only for UDF-formatted optical volumes.[1, 2]
WRKLNK	Yes	

Notes:

1. To perform this operation, you must be the owner of the file or have *ALLOBJ special authority.

   QOPT does not maintain or honor object level authorities associated with optical files and directories. Therefore, any attempt to change or revoke object level authorities is not allowed. The only allowed value for the New object authorities (OBJAUT) parameter is *SAME.

   You are not allowed to specify *EXCLUDE for the New data authorities (DTAAUT) parameter. Command parameter rules require that if *EXCLUDE is specified for the New data authorities parameter, a value of *NONE must be specified for the New object authorities parameter.

   If the desire is to revoke authority associated with the owner, group, or other user, *NONE may be specified as a value for the New data authorities parameter. In this case the specified user and the user's data authorities are removed from the list of authorized users.

   QOPT does not maintain or honor a private authority list. An attempt to assign New data authorities to a user other than the owner, group, or other (*PUBLIC) is not allowed.
2. QOPT does not maintain or honor a private authority list. An attempt to add a new user (option 1 from the WRKAUT display) and assign new data authorities to a user other than the owner, group, or other (*PUBLIC) is not allowed.

Option 4 is not supported to remove the user from list of authorized users. Select and prompt (F4) option 2 for the user you with to remove. The New data authorities parameter (DTAAUT) must be set to *NONE and the New object authorities parameter (OBJAUT) must be set to (*SAME).

Examples: Integrated file system

These programming examples demonstrate the use of the integrated file system UNIX-type APIs that pertain to the QOPT physical file system. The examples are written in Integrated Language Environment (ILE) C for the i5/OS operating system.

The programming examples demonstrate the following functions:

- Retrieving optical directory entries
- Creating an optical file
- Writing a file
- Closing a file
- Opening a file
- Reading a file
- Changing the offset into a file

Example code

This example program demonstrates the use of various integrated file system APIs. This program is written in C language.

Note: By using the following code examples, you agree to the terms of the "Code license and disclaimer information" on page 48.

```
/*********************************************************************/
/* This program demonstrates the use of various integrated file      */
/* system functions applied to the QOPT physical file system         */
/* including:                                    */
/*    chdir()   - change current directory                           */
/*    close()    - close file                                        */
/*    closedir()  - close directory                                  */
/*    creat()     - create file                                      */
/*    lseek()     - seek file (change file offset)                   */
/*    open()      - open file                                        */
/*    opendir()   - open directory                        */
/*    read()      - read file                              */
/*    readdir()   - read directory entry                   */
/*    rewinddir() - rewind directory entries                */
/*    stat()      - directory statistics                     */
/*    write()     - write file                               */
/*********************************************************************/
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <dirent.h>
#include <sys/stat.h>
#include <fcntl.h>

void main (void)
{
    /*****************************************************************/
    /* local variables, contents and defines                         */
    /*****************************************************************/
    char path[294];                     /* optical path              */
    DIR *dirP;                          /* pointer to the directory  */
    int filedes;                        /* open file descriptor      */
    struct dirent *direntP;             /* directory entry structure */
```

```
struct stat info;                   /* dir/file information      */
int volume_number;                  /* what it says...           */
int rc = 0;                         /* function return codes     */
int kk = 0;                         /* local counter             */
char data[] = "The quick red fox jumped over the fence";

/*****************************************************************/
/* Retrieve the list of volumes from the QOPT physical file      */
/* system by opening the QOPT pfs root directory and reading the */
/* directory entries.                                            */
/*****************************************************************/
memset(path,                        /* clear path name           */
0x00,
sizeof(path));
strcpy(path,                        /* set physical file system  */
"/QOPT");
rc = stat("/QOPT", &info);;         /* determine number of files */
if (rc != 0)
perror("stat() failed:");

dirP = opendir(path);               /* open the directory        */
if (dirP == NULL)
perror("opendir() failed:");

for (kk = 1; kk <= info.st_nlink; kk++)
{
direntP = readdir(dirP);
if (direntP == NULL)
perror("readdir() failed:");
printf("%d) %s\n", kk, direntP->d_name);
}

/*****************************************************************/
/* Prompt user for the volume they want to work with and make it */
/* the current directory.                                        */
/*****************************************************************/
printf("\nEnter the number the volume you want to work with:\n");
scanf("%d", &volume_number);;

rewinddir(dirP);                    /* beginning of directory    */
for (kk = 1; kk <= volume_number; kk++)
direntP = readdir(dirP);          /* get requested dir. entry    */

strcat(path, "/");
strcat(path, direntP->d_name);
rc = chdir(path);                   /* set current working dir.  */
if (rc != 0)
perror("chdir() failed:");
if (getcwd(path, sizeof(path)) == NULL)
perror("getcwd() failed:");
printf("\nThe current working directory is:  %s\n", path);

rc = closedir(dirP);                /* close the directory       */
if (rc != 0)
perror("closedir() failed:");

/*****************************************************************/
/* Create and open a file write only.  If the file exists it     */
/* will be truncated.  The owner will have read, write, and      */
/* execute authority to the file.                                */
/*****************************************************************/
strcat(path, "/");
printf("\nEnter a file name:\n");
scanf("%s", &path[strlen(path)]);

filedes = creat(path, S_IRWXU);
if (filedes == -1)
```

```
    {
    perror("creat() failed");
    return;
    }

    rc = write(filedes, data, sizeof(data));
    if (rc == -1)
    perror("write() failed:");

    close(filedes);

    /*****************************************************************/
    /* Read back the file and print it.                              */
    /*****************************************************************/
    memset(data, 0x00, sizeof(data));
    filedes = open(path, O_RDWR);
    if (filedes == -1)
    {
    perror("open() failed");
    return;
    }

    read(filedes, data, sizeof(data));
    if (filedes == -1)
    {
    perror("read() failed");
    return;
    }
    printf("\nThe data written to file is:  %s\n", data);

    /*****************************************************************/
    /* Change the offset into the file and change part of it.  Read  */
    /* the entire file, print it out and close the file.             */
    /*****************************************************************/
    lseek(filedes, 4, SEEK_SET);
    rc = write(filedes, "slow old ", 9);
    if (rc == -1)
    {
    perror("write() failed");
    return;
    }
    lseek(filedes, 18, SEEK_SET);
    rc = write(filedes, "went under ", 11);
    if (rc == -1)
    {
    perror("write() failed");
    return;
    }

    lseek(filedes, 0, SEEK_SET);
    read(filedes, data, sizeof(data));
    if (filedes == -1)
    {
    perror("read() failed");
    return;
    }
    printf("\nThe data now is:  %s\n", data);

    close(filedes);

    printf("Done...\n");
    return;

}
```

Related concepts

UNIX-Type APIs

Hierarchical file system programming

You can use the Hierarchical File System (HFS) APIs to read to or write from a directly attached or LAN-attached optical device. The HFS APIs are part of the i5/OS operating system.

The HFS API support for optical support consists of two parts:

- An application programming call interface to the hierarchical file system to manipulate objects known as files and directories.
- An optical or other registered file system that manages the storage devices where the files and directories are stored.

HFS API support includes the following optical functions:

- Creating or deleting a directory
- Opening, reading, or closing a directory
- Opening, reading, writing, or closing a file
- Locking or unlocking bytes in a file
- Getting or setting the size of a file
- Renaming, copying, deleting, or removing a file
- Retrieving or changing directory entry attributes

Hierarchical File System APIs

The use of Hierarchical File System (HFS) APIs is different for the optical file system, as compared to general API use.

Although the APIs that HFS supports are common to all file systems, each file system has different interpretations or restrictions regarding those APIs. The following table summarizes the optical interpretation of each HFS API. LAN-attached optical devices and directly attached optical devices have different restrictions for several of the APIs. Some examples of directly attached optical devices are CDs, DVDs, and SCSI attached optical libraries. Some examples of LAN-attached optical devices are Ethernet or token ring attached optical libraries.

Table 3. Optical HFS API restrictions

HFS APIs	Directly attached usage notes	LAN-attached usage notes
Change File Pointer (QHFCHGFP)	None.	None.
Close Stream File (QHFCLOSF)	None.	None.
Control File System (QHFCTLFS)	Supports the following requests: • SAV saves a held optical file. • RLS releases a held optical file. • SRD/VOL returns a sector read from an optical volume. • SRD/DEV returns a sector read from an optical device. • RTV/VOL returns volume-specific information. • GET reads file data directly from the media with minimal data caching. For UDF formatted volumes, GET requires *X authority to each directory in the path preceding the file and *R authority to the file.	Supports the following requests: • UPD/LAN performs a dynamic index refresh of the list of LAN volumes. • UPD/VOL returns volume-specific information. • RTV/VOL returns volume-specific information. • RTV/DIR returns subdirectory and file entries for a specified directory.

Table 3. Optical HFS API restrictions (continued)

HFS APIs	Directly attached usage notes	LAN-attached usage notes
Copy Stream File (QHFCPYSF)	If the source file is in the QOPT file system, *USE authority is required to the source optical volume. If the target file is in the QOPT file system, *CHANGE authority is required to the target optical volume. Copy information parameter, byte 1, option 2 is not supported (Copy Append). If specified, CPF1F62 will be returned. When the operation is complete, QCRTDTTM, QACCDTTM, and QWRTDTTM are set to the current date. When copying between the QOPT and QDLS file systems, file attributes are optionally copied depending on global optical attribute CPYATR. This attribute can be displayed or changed utilizing the CHGOPTA command. When copying between the QOPT and QDLS file systems, file permissions are not copied. If permissions need to be preserved between these file systems use the copy (CPY) CL command. If the source file is on a UDF formatted volume, *X authority is required to each directory in the path preceding the file. *R authority is required to the file. If the target file is on a UDF formatted volume, *WX authority is required to the parent directory and *X authority is required to each directory in the path preceding the parent directory.	If the source file is in the QOPT file system, *USE authority is required to the source optical volume. If the target file is in the QOPT file system, *CHANGE authority is required to the target optical volume. Copy information parameter, Byte 1, Option 2 is not supported (Copy Append). Copying from a volume in a directly attached library to a volume in a LAN-attached optical device is not supported.

Table 3. Optical HFS API restrictions (continued)

HFS APIs	Directly attached usage notes	LAN-attached usage notes
Create Directory (QHFCRTDR)	When the operation is complete, QCRTDTTM, QACCDTTM, QWRTDTTM are set to the current date. When the operation is complete, QFILSIZE and QALCSIZE are set to 0. Requires *CHANGE authority to the optical volume. Creating the optical root directory is not supported. Creating the volume portion of the directory is not supported. Attributes passed in the attribute information table are not supported, and will result in a CPF1F71 error message. The length of the attribute information table parameter must be 0. Optical attribute OPT.CHGATDTTM, which indicates the last time that the directory attributes were changed, is created. This date is set to the current date. If a user specifies an attribute, it is ignored. For UDF formatted volumes, *WX authority is required to the parent directory. *X authority is required to each directory in the path preceding the parent directory. The owner of the directory will be the user creating the directory and the owner data authorities will be set to *RWX. The primary group and primary group data authorities will be the same as the parent directory. The *PUBLIC data authorities will be the same as the parent directory.	When the operation is complete, QCRTDTTM, QACCDTTM, QWRTDTTM are set to the current date. When the operation is complete, QFILSIZE and QALCSIZE are set to 0. Requires *CHANGE authority to the optical volume. Creating the optical root directory is not supported. Creating a volume portion of a directory is not supported. All standard attributes are ignored. The length of attribute information table parameter must be set to 0.
Delete Directory (QHFDLTDR)	Deleting the optical root directory is not supported. Deleting the volume portion of a path is not supported. Requires *CHANGE authority to the optical volume. For UDF formatted volumes, *WX authority is required to the parent directory and *X authority is required to each directory in the path preceding the parent directory. *W authority is required to the directory being deleted.	Deleting the optical root directory is not supported. Deleting the volume portion of a path is not supported. Requires *CHANGE authority to the optical volume.

Table 3. Optical HFS API restrictions (continued)

HFS APIs	Directly attached usage notes	LAN-attached usage notes
Delete Stream File (QHFDLTSF)	Requires *CHANGE authority to the optical volume. For UDF formatted volumes, *WX authority is required to the parent directory. *X authority is required to each directory in the path preceding the parent directory. *W authority is required to the file being deleted.	Requires *CHANGE authority to the optical volume.
Get File Size (QHFGETSZ)	None	None
Set File Size (QHFSETSZ)	None	Not Supported

Table 3. Optical HFS API restrictions (continued)

HFS APIs	Directly attached usage notes	LAN-attached usage notes
Open Stream File (QHFOPNSF)	Parameter open information: • Opening with an access mode (byte 6) of write only or read/write requires *CHANGE authority to the volume. • Opening with an access mode (byte 6) of read only requires *USE authority to the volume. • Lock Modes (byte 5) are enforced across different open instances. If the same job opens a file multiple times, these open locks can conflict. If QALCSIZE was specified on an open request for the write operation, optical media will be checked to see if enough space is available. If not, error message CPF1F62 is returned. All standard attributes except QALCSIZE are ignored. If a file is being created, QCRTDTTM, QACCDTTM, and QWRTDTTM are set to the current date. If a file is being updated, QWRTDTTM is set to the current date. If a file is being read, no time stamps are changed. QACCDTTM is never changed after a file is created. It will always equal QCRTDTTM. The following authorization rules apply only for UDF formatted volumes: • If opening a file for READ, *X authority is required to each directory in the path preceding the file and *R authority is required to the file. • If opening an existing file for WRITE, *X authority is required to each directory in the path name preceding the file and *W authority is required to the file. • If opening an existing file for READ/WRITE, *X authority is required to each directory in the path name preceding the file and *RW authority is required to the file. • If creating the file, *WX authority is required to the parent directory. • If creating the file, the owner of the file will be the user creating the file and the owner data authorities will be set to *RWX. The primary group and primary group data authorities will be the same as the parent directory. The *PUBLIC data authorities will be the same as the parent directory.	Parameter Open information: • Byte 3 (write-through flag), is not supported. • Byte 7 (type of open operation to perform), is not supported. • Opening with an access mode (byte 6) of read-only requires *USE authority to the volume. Unless the file open attempt is for read-only access, attributes are not tolerated and result in error message CPF1F71. The length of the attribute information table parameter must be 0. If a file open attempt is for read-only access, attributes are tolerated but ignored.
Read Stream File (QHFRDSF)	None.	None.

Table 3. Optical HFS API restrictions (continued)

HFS APIs	Directly attached usage notes	LAN-attached usage notes
Retrieve Directory Entry Attributes (QHFRTVAT)	Requires *USE authority to an optical volume. For UDF formatted volumes, *X authority is required to each directory in the path name preceding the file and *R authority is required to the file or directory being read.	The user can retrieve only LAN-standard attributes: QFILSIZE, QCRTDTTM, and QWRTDTTM. Requires *USE authority to an optical volume. The length of attribute information table parameter must be set to 0.
Write Stream File (QHFWRTSF)	None.	None.
Change Directory Entry Attributes (QHFCHGAT)	QFILATTR is the only standard attribute that can be changed. All others that are specified are ignored. Read only flag, byte 1 of the QFILATTR attribute, can only be set for a file, not a directory. If specified for a directory, it is ignored. Changed flag, byte 5 of the QFILATTR attribute, can be set to either 0 or 1. It is automatically set on (1) whenever a file is created or written to. If OPT.CHGATDTTM is specified, it is ignored. Requires *CHANGE authority to an optical volume. For UDF formatted volumes, *X authority is required to each directory in the path name preceding the file and *W authority is required to the file.	API not supported.
Close Directory (QHFCLODR)	None.	API not supported.
Force Buffered Data (QHFFRCSF)	If the volume media format is *UDF, then data is forced to optical media. If the volume media format is not *UDF, then data is forced to internal disk storage, not to optical media. For a file opened for read-only access, this API has no effect.	API not supported.
Lock and Unlock Range in Stream File (QHFLULSF)	None.	API not supported.

Table 3. Optical HFS API restrictions (continued)

HFS APIs	Directly attached usage notes	LAN-attached usage notes
Move Stream File (QHFMOVSF)	If the source file is in the QOPT file system, *CHANGE authority is required to the optical source volume. If the target file is in the QOPT file system, *CHANGE authority is required to the optical target volume. When moving between the QOPT and QDLS file systems, file attributes are optionally copied depending on the global optical attribute CPYATR. This attribute can be displayed or changed using the CHGOPTA command. If the source file is on a UDF formatted volume, *WX authority is required to the parent directory and *X authority is required to each directory in the path name preceding the parent directory. *RW authority is required to the file. If the target file is on a UDF formatted volume, *WX authority is required to the parent directory and *X authority is required to each directory in the path name preceding the file.	API not supported.
Open Directory (QHFOPNDR)	Opening the file system root (/QOPT) will allow both directly attached and LAN-attached volumes to be returned on Read Directory Entries. Lock mode is ignored when opening the file system root. Lock mode of no lock is not supported. If requested, a lock mode of deny none is substituted. Requires *USE authority to the optical volume. For UDF formatted volumes, *X authority is required to each directory in the path name preceding the directory being opened and *R authority is required to the directory being opened.	API not supported.
Read Directory Entries (QHFRDDR)	QNAME is returned without the QOPT file system name. QNAME is the only field that is set for a LAN-attached volume. QWRTDTTM will always equal QCRTDTTM. For files and directories, QACCDTTM will always equal QCRTDTTM. For volumes, QACCDTTM will equal the last volume reference date.	API not supported.

Table 3. Optical HFS API restrictions (continued)

HFS APIs	Directly attached usage notes	LAN-attached usage notes
Rename Stream File (QHFRNMSF)	Requires *CHANGE authority to the optical volume. For UDF formatted volumes, *WX authority is required to the parent directory and *X authority is required to each directory in the path name preceding the parent directory. *W authority is required to the file being renamed.	API not supported.
Rename Directory (QHFRNMDR)	API not supported.	API not supported.

Control file system functions

The Control File System (QHFCTLFS) API enables optical support to perform unique operations for the optical file system.

The following functions are optical-specific functions that are not otherwise available through the HFS APIs. Different functions are available for directly attached and LAN-attached optical devices.

Control file system functions for directly attached optical devices

These control file system functions are available for directly attached optical devices:

- **SAV**. Saves a held optical file.
- **RLS**. Releases a held optical file.
- **SRD/VOL**. Performs a sector read to an optical volume.
- **SRD/DEV**. Performs a sector read to an optical device.
- **RTV/VOL**. Returns volume-specific information.
- **GET**. Reads file data directly from the media with minimal caching.

Save held optical file function

Use the QHFCTLFS API to save a held optical file. A process must have read access to a held optical file to save it.

Here is the syntax for the input buffer for the QHFCTLFS program:

```
'SAV' + '/' + held-file-path + '//' + destination-file-path
```

For example:

- Input data buffer: `SAV/VOLUME1/DIRECTORY1/FILE1//VOLUME2/DIRECTORY2/FILE2`
- Input data buffer length: `54`

This function is also available using an option on the Work with Held Optical File (WRKHLDOPTF) display. However, unlike the save option on the Work with Held Optical File (WRKHLDOPTF) display, the save held optical file function of the control file system API does not automatically release a held file after it is saved. Therefore, an explicit release held optical file request is needed afterward.

Release held optical file function

The QHFCTLFS API clears the held status of a file and releases the optical file system from its obligation to write to the optical disk. A process must have read and write access to a held file in order to release it. This means that no locks can currently be imposed on the file by other active jobs.

Here is the syntax for the input buffer for the QHFCTLFS program:

```
'RLS' + '/' + held-file-path
```

For example:

- Input data buffer: `RLS/VOLUME1/DIRECTORY1/FILE1`
- Input data buffer length: `28`

This function is also available using an option on the Work Held Optical File (WRKHLDOPTF) display.

Sector read function

The QHFCTLFS API performs a sector reading of optical media. The sector read function is useful if the application knows precisely where data is stored on the optical media. Sector read functions can be accomplished without opening and closing files and independently of all HFS APIs. Multiple sectors can be read at one time.

There are two variations of the input buffer for issuing the Control File System sector read function:

```
SRD/VOL/volume_name/starting sector/number of sectors
SRD/DEV/device_name/starting sector/number of sectors
```

Both return the range of sectors requested by the user. Sectors can be requested from an optical volume or optical device. For example, if an application wanted to read five sectors of optical volume `VOL01` beginning at sector 1000, `SRD/VOL/VOL01/1000/5` is requested.

Note: DEV is valid for stand-alone CD and DVD devices.

Retrieve volume information function

The QHFCTLFS API retrieves information about a particular volume.

Here is the input buffer format for the QHFCTLFS program:

```
RTV/VOL/volume_name
```

The format of the information returned in the output buffer is identical to the output file structure for volume attributes (QAMODVA).

Get file data function

The QHFCTLFS HFS API reads a block of data from a file directly into your output buffer. This function improves performance when reading an entire file sequentially or when reading large blocks of data. The optical file system will not copy or cache the data as it does through normal Open, Read, and Close Stream File HFS APIs. When doing random read operations to a file, the Open, Read, and Close Steam File APIs still provides the best performance.

The following restrictions apply when using this API:

- Align output buffer on a 512-byte boundary.
- Maximum-read size is 16 384 000 bytes.
- The HFS API requires Shared No Update (*SHRNUP) access to the file.

- Calling program must be in user (not system) state.
- The HFS API requires *USE authority to the volume.

Here is the syntax for the input buffer for the QHFCTLFS program:

```
'GET' + '/' + entire path + '//' + bytes to read + '/' + file offset
```

The following example will read 15 MB from FILE.XXX, starting at the beginning of the file with (offset=0):

- Input data buffer: GET/VOL1/DIR1/SUBDIR1/FILE.XXX//15728640/0
- Input data buffer length: 42

The number of bytes read is returned in the Length of data returned parameter. In the above example if FILE.XXX is only 50 KB in size, 51200 will be returned in the field. Therefore, it is not necessary to know the file size before issuing this request. Likewise, if 15728640 is returned in the Length of data returned parameter, the file is at least 15 MB in size. More read operations may be necessary to retrieve all the data.

It is not required that the number of bytes to read be a multiple of 4096. However, if the number is not a multiple of 4096, data may be read into the output buffer beyond the number of bytes requested. This is because the device does I/O in blocks of 4096 bytes. Therefore, reading data in multiples of 4096 bytes is advised in order to avoid this problem.

Errors from control file system (GET function)

The following table shows some common application errors that may occur using this API.

Table 4. Common errors for the GET function

Message	Error
OPT1812 with 6030 as unexpected return code	File offset is beyond the end of file.
OPT1812 with A950 as unexpected return code	Output buffer is not 512-byte aligned.
OPT1860	Bytes to read is greater than the buffer size.
OPT1812 with C060 as unexpected return code	Attempted to read more than 16 384 000 bytes.
CPF1F48	Input buffer is not valid. Verify the syntax.

Control file system functions for LAN-attached optical devices

The following control file system functions are available for LAN-attached media libraries.

- UPD/LAN - performs a dynamic refresh of the LAN volume lists.
- UPD/VOL - returns volume-specific information.
- RTV/VOL - returns volume-specific information.
- RTV/DIR - returns subdirectory and file entries for a specified directory.

Update volume information function

The QHFCTLFS API retrieves information about a particular volume or updates the internal list of available volumes on a LAN.

Here is the input buffer format for the QHFCTLFS program:

```
UPD/VOL/volume_name
```

It performs as follows:

- UPD/VOL/volume-name: Using this input buffer format returns the amount of free space on a volume, total user space, media type, and opposite-side volume ID. The format is shown here:
  - Bytes (1-32): Opposite-side volume ID.
  - Bytes (33): Reserved.
  - Bytes (34-37): User free space on the volume. This is a 4-byte binary field.
  - Bytes (38-41): Total free space on the volume. This consists of the user free space on the volume plus the reserved space on the volume. The reserved space on the volume is determined when setting the volume-full threshold for the volume. This is a 4-byte binary field.
  - Bytes (42): Media type. This is a 1-byte binary field that can have the following values.
    - 0 = Nonvalid Media or 3431 Standalone Drive
    - 1 = Write Once Read Many (WORM) media
    - 2 = Rewriteable media
  - Bytes (43): Magnitude of free space on the volume. This is a 1-byte binary field that can have the following values:
    - 0 = Space field is in number of bytes.
    - 1 = Space field is in number of kilobytes (1024).
    - 2 = Space field is in number of megabytes (1048576).
  - Bytes (44): Magnitude of Total Space on the Volume. This is a 1-byte binary field that can have the following values:
    - 0 = Space field is in number of bytes.
    - 1 = Space field is in number of kilobytes (1024).
    - 2 = Space field is in number of megabytes (1048576).
- UPD/LAN: Using this input buffer format updates an internal list of available volumes on all activated servers. You can perform this function after adding or removing cartridges from data servers.

Retrieve volume information function

The QHFCTLFS API retrieves information about a particular volume.

Here is the input buffer format for the QHFCTLFS program:

```
RTV/VOL/volume_name
```

The format of the information returned in the output buffer is identical to the output file structure for volume attributes (QAMODVA).

The system uses format QAMODVA for volumes in all optical device types. While the format is the same, not all fields contain a value for LAN volumes.

Retrieve directory information function

The QHFCTLFS API retrieves a list of files and subdirectories for a particular directory.

Here is the input buffer for the QHFCTLFS program:

```
RTV/DIR/volume_name/directory_name
```

The directory information is returned in the output buffer in the following format:

- CBdirectoryBCBdirectoryBCBfilenameBCBfilenameBB, whereas:
  - C
    - D = Directory entry
    - F = File name entry

– B = EBCDIC blank
– BB = Double EBCDIC blanks to indicate end of string

The output buffer must be at least 31 KB long.

Standard attributes

Directory entries for files and directories have information that is associated with them called *attributes*. Each attribute consists of a name and a value. *Standard attributes* are those attributes that generate automatically when you create a directory or file. Standard attribute names start with the letter Q for ease of identification.

All file systems use standard attributes. Several receive unique interpretation by the optical file system. LAN-attached optical devices have a different interpretation of standard attributes than directly attached optical devices.

QALCSIZE attribute

As an output field, QALCSIZE is the number of bytes allocated on optical disk by the file. It will always be 0 for directories.

When the QALCSIZE attribute is specified on the Open Stream File during a write request, the media is checked to see if there is enough space available to allocate the amount specified. If there is not enough space available on the optical volume, message CPF1F61, **No free space available on media**, is issued. "Media capacity and volume threshold" on page 34 contains more information about using this attribute.

QACCDTTM attribute

The QACCDTTM attribute is not supported by the optical file system. It is always the same as the file creation date and time (QCRTDTTM) attribute.

QCRTDTTM attribute

The QCRTDTTM attribute indicates the creation date of a file or directory.

QWRTDTTM attribute

The QWRTDTTM attribute indicates the last date and time that data was written to an optical file. It does not reflect the date and time when the file attributes were last written.

QFILATTR attribute

Support of the QFILATTR attribute is only provided by directly attached optical support devices. The optical interpretation of the file flags is as follows:

- **Read-only file**: The i5/OS operating system provides full support of this attribute through the optical file system. When setting this attribute to ON (1), you cannot delete or overwrite the file.
- **Hidden file**: The i5/OS operating system maintains this attribute for the user application to manage, but does not fully support it through the optical file system. When setting this attribute to ON (1), the optical file system does not recognize the file as hidden. User applications require no special access to files that have this attribute set on.
- **System file**: The i5/OS operating system maintains this attribute for the user application to manage, but does not fully support it through the optical file system. When setting this attribute to ON (1), the optical file system does not recognize the file as a system file. User applications require no special access to files that have this attribute set on.

- **Changed file**: The i5/OS operating system supports this attribute by the optical file system. It is automatically set to ON (1) when a file is created or written to. You can only set it to OFF (0) by using the Change Directory Entry Attributes (QHFCHGAT) API.

Extended attributes

Extended attributes are special attributes for files and directories that are not standard and therefore not recognized by the hierarchical file system (HFS). They are typically defined by a business application, but some are recognized by the optical file system as having special meanings.

OPT.CHGATDTTM attribute

The OPT.CHGATDTTM attribute reflects the last date and time that the file attributes were written. It is returned to the user application as an extended attribute through the Retrieve Directory Entry Attributes (QHFRTVAT) command.

QOPT.IOMETH attribute

The QOPT.IOMETH attribute is a special attribute to the optical file system. Support is provided only by directly attached optical support devices. It is ignored by LAN support. The system also ignores this attribute when the media format is Universal Disk Format.

When an extended attribute of this name is passed by the application as the attribute name field in the Attribute Information Table (AIT) during an open stream file request, the optical file system knows that a special method of I/O is being requested. The optical file system retrieves the special method of I/O from the attribute value field in the AIT.

Currently, there is only one special method of I/O supported by the optical file system: You can request this method of I/O when the attribute value field for the QOPT.IOMETH attribute contains the value (EXPNBUFF). The optical software recognizes this special extended attribute as a requested I/O method, and not as a normal extended attribute. It is not hereafter associated with the file in any way, and does not appear when attributes for the file are retrieved. All read operations for the process use expanding buffer I/O until the file is closed. Methodology and restrictions for using expanding buffer I/O are listed here. In order to determine if expanding buffer I/O should be used, see the Expanding buffer I/O method topic.

An HFS attribute in an attribute information table consists of several fields. These fields and the values you specify when opening a file for expanding buffer I/O are summarized in the following table.

Table 5. Expanding buffer attribute definition

Field	Data type (see note)	Value for EBIO
Attribute name	CHAR(*)	QOPT.IOMETH
Attribute value	CHAR(*)	EXPNBUFF
Length attribute name	BIN(4)	0000000B
Length attribute value	BIN(4)	00000008
Notes: • CHAR(*) indicates a variable number of bytes of character information. • BIN(4) indicates 4 bytes of binary information. • All character fields should be set in uppercase.		

In addition to the values for attribute fields, two additional fields are required to build an attribute information table:

- The number of attributes defined in the table

- The table offset to each attribute, in bytes

The Open Stream File (QHFOPNSF) API requires 10 bytes of open information as input. When you attempt to open a file for expanding buffer I/O the open information is subject to the following restrictions:

- The action to take if a file exists must be to open the file.
- The action to take if a file does not exist must be to return an error.
- The lock mode for the file must be Deny Write or Deny Read/Write (exclusive).
- The access mode for the file must be Read Only.

If there is an expanded buffer I/O attribute in the attribute information table and any of these restrictions are not observed, an OPT1133 message is issued, indicating which of the fields in the open information was passed in error.

The APIs topic contains more information about the format of attributes, the Attribute Information Table, or the Open Stream File API.

Restrictions for expanding buffer I/O

In addition to the restrictions that are detailed when opening a file for expanded buffer I/O, you cannot use the following APIs for expanding buffer I/O, after a file is opened:

- Lock or Unlock Range in Stream File
- Set Stream File Size
- Write Stream File

Related concepts

Application programming interfaces

Related tasks

"Expanding buffer I/O method" on page 35
When you use the QOPT.IOMETH extended attribute to open a stream file through the hierarchical file system (HFS), you can improve performance for applications that typically read portions, but not all, of the data in large optical files. This method of input/output is referred to as *expanding buffer I/O*.

Copied file attributes using hierarchical file system

File attributes can be copied between file systems that support the Hierarchical File System (HFS) APIs.

Copied attributes between QOPT and QDLS file systems

When you copy files between QOPT and QDLS file systems using the hierarchical file system, the target file is assigned either default file attributes or the file attributes of the source file. This depends on the value you specify for the copy attributes (CPYATR) global value on the Change Optical Attributes (CHGOPTA) command.

When the CPYATR global value is specified as *NO on the CHGOPTA command, default file attributes are created for files that are copied between the QOPT and QDLS file systems.

When the CPYATR global value is specified as *YES on the CHGOPTA command, file attributes from the source file are copied to the target file for copies between the QOPT and QDLS file system.

Copied attributes from QDLS to QOPT

In a copy operation or move operation from QDLS to QOPT, the following default attributes are assigned to the target file:

- Standard file attributes:
  - Creation date and time is set to the current date and time.
  - Modification date and time is set to the current date and time.
  - Access date and time is set to the current date and time.
  - The QFILATTR standard attribute is set to `00000`; the file is not read-only, the file is not hidden, the file is not a system file, the file is not a directory, and the file has not changed since it was last archived or created.
- No DIA document attributes are copied.
- No user-defined extended attributes are copied.

The file name (QNAME) and file size (QFILSIZE) are maintained.

Copied attributes from QOPT to QDLS

In a copy operation or move operation from QOPT to QDLS, the following default attributes are created:
- Standard file attributes:
  - Creation date and time is set to the current date and time.
  - Modification date and time is set to the current date and time.
  - Access date and time is set to the current date and time.
  - The QFILATTR standard attribute is set to `00000`; the file is not read-only, the file is not hidden, the file is not a system file, the file is not a directory, and the file has not changed since it was last archived or created.
- DIA document attributes:
  - DIA.CA04C700 (text description) is set to the file name.
  - DIA.CA04C701 (profile GCID) is set to code page 697 and character set 500.
  - DIA.CA04C706 (file type) is set to `000E` (PC file).
  - DIA.CA04C720 (library assigned document name) is assigned to represent this file.
  - DIA.CA04C708 (last changed date and time) is set to the current date and time.
  - DIA.CA04C707 (creation date and time) is set to the current date and time.
  - DIA.CA04C710 (NLS information) is set to the language ID and country or region ID of the job.
  - DIA.CA04C740 (file date and time) is set to the current date and time.
- No user-defined extended attributes are copied.

The file name (QNAME) and file size (QFILSIZE) are maintained.

Example: Programming Hierarchical File System APIs for the optical file system

These hierarchical file system (HFS) examples can help you program your optical file system.

This topic demonstrates how the HFS API can be used with the ILE RPG programming language.

The programming examples demonstrate the following functions:
- Retrieving a path name from an array
- Calling the HFS API to open a stream file
- Calling the HFS API to write a 256-byte buffer passed to the program as a parameter
- Calling the HFS API to close the stream file

Note: By using the following code examples, you agree to the terms of the "Code license and disclaimer information" on page 48.

Getting a path and calling subroutines

This example gets a path and calls subroutines.

```
E                     AR     1   5 36
C        *ENTRY       PLIST
* 2 PARAMETERS - A DATA BUFFER ID AND AN INDEX TO THE ARRAY
C                 PARM           DATAIN    256
C                  PARM           IDX        10
* MOVE THE ARRAY ELEMENT TO A FIELD CALLED "PATH"
C                  MOVE AR,IDX    PATH
* EXECUTE SUBROUTINES TO OPEN, WRITE AND CLOSE A FILE
C                  EXSR OPNSF
C    RTCD        IFEQ 0
C                 EXSR WRTSF
C                  EXSR CLOSF
C                  END
C                  SETON                     LR
* TABLE/ARRAY . . . . . . . :   AR
** /QOPT/MYVOL1/DIRA/FILE
/QOPT/MYVOL1/DIRA/SUBDIRB/FILE
/QOPT/MYVOL1/DIRA/SUBDIRB/C/FILE
/QOPT/MYVOL1/DIRA/SUBDIRB/C/D/FILE
/QOPT/MYVOL1/DIRA/SUBDIRB/C/D/E/FILE
```

Defining data structures for opening files

This example defines data structures in the HFS.

```
* PATH LENGTH PARAMETER
IPATHLN     DS
I                                B   1    40PATHL
* OPEN INFORMATION PARAMETER
IOPNINF     DS
I                                1   1 EXISTS
I                                2   2 NOTTHR
I                                3   3 SYNASY
I                                4   4 RSV1
I                                5   5 SHAREM
I                                6   6 ACCESS
I                                 7   7 OTYPE
I                                8  10 RSV3
* ATTRIBUTE LENGTH PARAMETER
IATTRLN    DS
I                                B   1    40ATTRL
* RETURN CODE PARAMETER
IRETCD      DS
I                                B   1    40RCLEN
I                                B   5     80RTCD
I                               9   15    CONDTN
I                                16  16       RSV
I                                 17  272      MSG
* BYTES TO READ/WRITE PARAMETER
IBYTRDW       DS                  B   1   40BYT2RW
* BYTES ACTUALLY READ/WRITTEN PARAMETER
IBYTACT       DS                B   1   40BYTARW
```

Opening an optical file

This example opens an optical file.

```
* PARAMETER LIST FOR QHFOPNSF CALL
C           POPNSF    PLIST
C                     PARM             FHDLE  16
C                     PARM            PATH   36
C                     PARM             PATHL
C                     PARM             OPNINF
```

```
C                     PARM             ATRTBL  1
C                     PARM             ATTRLN
C                     PARM             ACTION  1
C                     PARM             RETCD
C* OPEN FILE SUBROUTINE
C           OPNSF     BEGSR
C* FILL IN THE PATH AND ATTRIBUTE LENGTHS
C                    Z-ADD36          PATHL  SET PATH LEN=36
C                     Z-ADD*ZEROS      ATTRL  ZERO ATTRIBUTE LENGTH
C* FILL IN THE OPNINF PARAMETER
C                     MOVE '0'         EXISTS   1   FAIL IF EXISTS
C                     MOVE '1'         NOTTHR   1  CREATE IF NOT THERE
C                     MOVE '0'         SYNASY   1  ASYNCHRONOUS
C                     MOVE *BLANKS     RSV1     1
C                     MOVE '1'         SHAREM   1  DENY NONE
C                    MOVE '2'         ACCESS   1  READ/WRITE
C                     MOVE '0'        OTYPE    1  NORMAL
C                     MOVE *BLANKS     RSV3     3
C* CALL THE API TO OPEN THE STREAM FILE
C                     CALL 'QHFOPNSF'POPNSF         50
C           OPNEND    ENDSR
```

Writing a file to an optical disk

This example writes a file to an optical disk.

```
* PARAMETER LIST FOR QHFRDSF OR QHFWRTSF CALL
C           PRWSF     PLIST
C                     PARM           FHDLE   16
C                     PARM           DATAIN
C                     PARM           BYT2RW
C                     PARM           BYTARW
C                     PARM           RETCD
C* CALL API TO WRITE TO THE FILE
C           WRTSF     BEGSR
C                    Z-ADD256       BYT2RW SET WRITE LENGTH=256
C                    CALL 'QHFWRTSF'PRWSF          50
C           WRTEND    ENDSR
```

Closing an optical file

This example closes an optical file.

```
* PARAMETER LIST FOR QHFCLOSF CALL
C           PCLOSF    PLIST
C                 PARM            FHDLE   16
C                 PARM           RETCD
C* CALL API TO CLOSE THE FILE
C           CLOSF     BEGSR
C           CALL 'QHFCLOSF'PCLOSF         50
C           CLSEND    ENDSR
C* END OF SAMPLE RPG CALL TO THE HFS API
```

Related concepts

Application programming interfaces

Tips: Optical device programming

The techniques in this topic are often helpful in designing custom optical programs for your business.

This topic describes how the optical file system manages file data so application programmers can optimize their applications. Since applications have different requirements, this topic does not suggest the best way to write an optical application. It does, however, provide explanations that all application programmers can find useful.

Use this topic to determine the best way to handle optical file management, either through the HFS or UNIX-type APIs. Use this topic only for applications to directly attached optical support.

Note: Concepts in this topic do not apply to optical LAN support.

Media capacity and volume threshold

The optical file system provides a logical threshold capability to prevent applications from reaching the absolute volume capacity. The logical threshold is defined when the volume is initialized, and is unique for each volume. You can change this threshold by using the Change Optical Volume (CHGOPTVOL) command.

Before you begin

Note: The logical volume threshold is applicable only for the high performance optical file system (HPOFS) media format. For Universal Disk Format (UDF) media format, the logical volume threshold is always 100% and cannot be changed.

You need to devise a strategy to deal with the situation when the media becomes full. This is especially true when writing to Write Once Read Many (WORM) media. You might consider the following questions:

- How should I use the volume threshold?
- What should I do when the volume is full?
- How can I prepare for a volume-full condition?

The logical volume threshold is applicable only for the HPOFS media format. For UDF media format, the logical volume threshold is always 100% and cannot be changed.

The volume threshold is provided to allow applications to prepare for an actual volume-full condition. When WORM media becomes full, there can be no further write operations. Depending on the requirements of the application, the threshold can be used in various ways to prepare for the media becoming physically full.

For example, an application might write groups of spooled files to optical disk. After each group is written, an additional file might be written that contains an index to the spooled files just written. Without the index, the spooled files can be useless. Unless the application can manage the media capacity, the volume might run out of space before the index file can be written. One way to avoid running out of space is to set the volume threshold to 99%. When the message `No space available` is issued, the application can then increase the threshold to 100% and write any necessary additional files.

Media capacity management on a per-file basis

An application might need to manage the media capacity on a per-file basis.

Before you begin

The following methods help you determine if a file fits on a volume.

- Handle error on a close operation.

  Assume an optical volume is initialized to a 95% threshold and an application writes files until the volume threshold is reached. When the threshold is reached, the application will receive message CPF1F61, `No free space available on media`. At this point, the volume threshold can be increased to 97% (or anything else up to 100%) by using the CHGOPTVOL command. You can then attempt to close the file.
- Specify QALCSIZE on the Open Stream File HFS API.

Another method to determine if a file will fit on a volume is by specifying an allocation size (QALCSIZE) on an open stream file. On an open stream file, the system can pass a value in attribute QALCSIZE. This attribute is valid when the open operation is for create or replace; otherwise, it is ignored. Specifying a value for QALCSIZE results in comparing the specified value against the space available on the volume. If the space available is less than QALCSIZE, then the system issues message CPF1F61. The space available must exceed the QALCSIZE in order for the open operation to occur. Only on the first open instance of a file honors this attribute. If specified by more than one opening of a file, the system ignores the additional attributes.

Note: This does not actually allocate space on the optical volume at the time of the open operation. It checks the volume to see if the number of requested bytes are available.

There are drawbacks to using this method:

1. You need to know the size of the file you are creating at the time you make the open request.
2. If multiple jobs are writing to the same media, there is no guarantee that by the time the data is written, the space will still be available.

If the size of the file is known before the time the open request is made, and there will not be other jobs writing to that volume during the time your file is open, this is an excellent method to check media capacity before creating a file.

- Retrieve the space available on a volume.

  Another method is to have the application retrieve the space available on the volume. You can do this by using the Display Optical (DSPOPT) command through output file support. The output file can then be read to retrieve the number of bytes that are assumed to be available on the media.

Expanding buffer I/O method

When you use the QOPT.IOMETH extended attribute to open a stream file through the hierarchical file system (HFS), you can improve performance for applications that typically read portions, but not all, of the data in large optical files. This method of input/output is referred to as *expanding buffer I/O*.

Before you begin

Expanding buffer I/O is available only to HFS API applications when accessing high performance optical file system (HPOFS) or ISO 9660 formatted media. This attribute is ignored when the media format is UDF.

Note: Using the HFS APIs, optical file data is buffered into a virtual optical file in i5/OS main storage. If expanding buffer I/O is not selected as an option, the size of this buffer is equal to the size of the actual optical file. For example, a 100 MB file on optical media has a 100 MB buffer when the file is opened through the HFS API Open Stream File. The performance cost for overhead operations involving the optical buffer is proportional to the buffer size. The time it would take to read one byte of a 100 MB file is substantially greater than reading one byte of a 50 KB file.

When an optical file is opened for expanding buffer I/O, the size of the buffer begins at zero and expands as data is read into the buffer as requested by the application. The minimum amount of the size expansion is 256 KB. The buffer expands only if the requested data is contained within a logical 256 KB page that is not yet contained in the buffer. For these reasons, the amount of time it would take to read one byte of a 100 MB file opened for expanding buffer I/O should be roughly identical to the time to read one byte of a 50 KB file opened in the same manner.

About this task

Situations in which expanding buffer I/O is useful

Expanding buffer I/O should be considered as an option for improving the performance of the read operation if any of the following conditions are met:

- The typical size of an optical file to be read is greater than 256 KB.
- The amount of data read from the optical file between the open and close stream file is a fraction of the total file data. The exact fraction would be impossible to specify, but the performance improvements that are achieved will be greater the smaller the fraction. For example, an application that used expanding buffer I/O to read 25 KB of a 50 MB file would experience much greater performance improvements than an application that read 45 MB of the same file. An application that reads the entire 50 MB example file 40 KB at a time through multiple reads probably would not experience any performance improvement using expanding buffer I/O.
- The application will not issue the Set Stream File Size, Lock-Unlock Byte Range, or Write Stream file APIs while the file is open for expanding buffer I/O.

Related reference

"Extended attributes" on page 29
Extended attributes are special attributes for files and directories that are not standard and therefore not recognized by the hierarchical file system (HFS). They are typically defined by a business application, but some are recognized by the optical file system as having special meanings.

Forced buffered data APIs

Forced buffered data APIs synchronously force file and directory information. When you use the Force Buffered Data (QHFFRCSF) or fsync() API, you can write optical file data to nonvolatile media while you are writing optical file data to optical media.

Before you begin

When creating or updating optical files, the data is not guaranteed to exist on optical disk until the file is successfully closed. Optical file data can, however, be synchronously written to nonvolatile storage using either the QHFFRCSF or fsync() API. The type of nonvolatile storage is different depending on the optical media format.

For the high performance optical file system (HPOFS), all file data will be written to the internal disk storage. The data can then be recovered through a held optical file if a power loss or other unexpected error occurred which prevented the file from being closed.

For Universal Disk Format (UDF), all file data is written to the optical disk when a force operation is issued. No recovery is required if a power loss or other unexpected error occurs that prevents the file from being closed. However, if write operations are issued after the data is forced and the close operation is not successful, the file data is unpredictable. Because the write operations that follow the force operation are asynchronous, the data might not be written to the optical disc.

Management of held optical files

Virtual files that are held due to an error while writing to optical media can be saved to another volume. *Held optical files* are virtual files that were never successfully written to optical media.

Before you begin

A virtual file becomes *held* if an error occurs during the close operation of a file on a non-UDF formatted volume. You can manage these virtual files by using application interfaces and optical utilities. No creation of held files occurs for files that fail to archive on UDF formatted volumes.

Assume an optical volume is initialized to a 95% threshold and an application writes files until the volume threshold is reached. When the threshold is reached, the application will receive message CPF1F61, `No free space available on media`. In this example, the absolute volume capacity is reached and the file is too large to fit on the volume. Because increasing the volume threshold will not help, another solution is needed. When the close request fails, the virtual file becomes held. Using the Work With Held Optical Files command, this virtual file can be saved to another volume. If you want, the file

can be saved under a different name. The save request can also be performed using a control file system function.

Path names requirements

The term *path name* refers to a file-system name, volume name, directory name, and file name.

Path names for volumes in directly attached devices

In the path name for volumes in directly attached devices, the forward slash (/) is used as a separator character. The path name must begin with a forward slash and contain no more than 294 characters. See the following example for the format of a path name on a directly attached device:

```
/QOPT/VOL_NAME/DIRECTORY_NAME/SUB_DIR1/.../SUB_DIRn/FILE_NAME
```

QOPT refers to the optical file system. You must use it to qualify the optical file system when issuing calls to optical support through the HFS API or the Unix-type APIs. The portion of the path following the file system name cannot contain more than 289 characters. For the rules for using path names, see the following items:

- A path name can consist of any EBCDIC characters, except the characters that are listed below:
  - X'00' through X'3F'
  - X'FF'
  - The quotation mark (")
  - The asterisk (*)
  - The less than (<) and greater than (>) signs
  - The question mark (?)
  - The hyphen (-)
  - The back slash (\)

  When accessing UDF formatted volumes through the integrated file system APIs, the only characters not valid are X'00' through X'3F', X'FF', and back slash.
- The volume identifier can be a maximum of 32 characters for HPOFS media format, and a maximum of 30 characters for UDF media format. The identifier must contain only alphabetic characters (A through Z), numeric characters (0 through 9), a hyphen (-), an underscore(_), or a period (.). The first character must be alphabetic or numeric, and the identifier cannot contain blanks.
- You can include one or more directories in the path name, but it is not required. The total number of characters in all of the subdirectories together cannot exceed 256 characters.
- The file name is the last element in the path. The directory length in the path limits the file name length. The directory name and file name combined cannot exceed 256 characters. The preceding forward slash of the directory name is considered part of this 256 characters.

Path names for volumes in LAN-attached devices

For a path name on an optical volume in a LAN-attached optical device, the forward slash (/) is used as a separator character. The path name must begin with a forward slash and contain no more than 261 characters. See the following example for the format of a path name on an optical volume in a LAN-attached optical device:

```
/QOPT/VOL_NAME/DIRECTORY_NAME/SUB_DIR1/.../SUB_DIRn/FILE_NAME
```

QOPT refers to the optical file system, and must be used to qualify the optical file system when issuing calls to optical support through the HFS or integrated file system APIs. The portion of the path following the file system name cannot contain more than 256 characters. For the rules for using path names on LAN-attached devices, see the following items:

- See IBM 3995 Optical Library Dataserver Operator Guide for C-Series Models  for the allowed characters for path names.
- The volume name is required and can contain a maximum of 32 characters.
- One or more directories can be included in the path name, but it is not required. The total number of characters in all of the subdirectories together cannot exceed 254 characters.
- The file name is the last element in the path. The file name length is limited by the volume and directory length in the path. The volume name, directory name, and file name combined cannot exceed 256 characters. The preceding forward slashes of the volume and directory name are considered part of the 256 characters.

Examples: Moving spooled files to and from optical storage

These basic optical programming examples use APIs to create control language (CL) programs.

Copy Stream File: Command source

Note: By using the following code examples, you agree to the terms of the "Code license and disclaimer information" on page 48.

```
/*********************************************************************/
/*                            */
/* COMMAND NAME:   CPYSTRF                                           */
/*                                  */
/* COMMAND TITLE: Copy Stream File                                   */
/*                                      */
/* COMMAND DESCRIPTION: Copy stream file between two file systems    */
/*                                        */
/*********************************************************************/
         CMD  PROMPT('Copy Stream File')

           PARM      KWD(SRCFILE) TYPE(*CHAR) LEN(300) MIN(1) +
        MAX(1) PROMPT('Source file name')        +
                    VARY(*YES)
    PARM      KWD(TGTFILE) TYPE(*CHAR) LEN(300) MIN(1) +
         MAX(1) PROMPT('Target file name')        +
                    VARY(*YES)
    PARM     KWD(RPLFILE)TYPE(*CHAR) LEN(6) DFT(*NO) +
                    SPCVAL((*NO '0     ') (*YES '1     '))   +
                    PROMPT('Replace existing file')
```

Copy Stream File: CL program source

This CL example can be used to copy stream files between file systems.

```
/*********************************************************************/
/*                            */
/*  PROGRAM: CPYSTRF (Copy Stream File)                              */
/*                                                                   */
/*                                                                   */
/*  DESCRIPTION:                                                     */
/*  This is the CL program for sample CL command CPYSTRF.  This      */
/*  program can be used to copy stream files between file            */
/*  systems.  The actual copy is done by making a call to            */
/*  the HFS API program QHFCPYSF (Copy stream file).                  */
/*                                                                   */
/*                                                                  */
/*  INPUT PARAMETERS:                                                */
/*  - Complete source path                                            */
/*    Example: /filesystem/directory1/directoryx/file                */
/*             /QDLS/DIRA/DIRB/FILE01                                */
/*                  - or -                                    */
/*             /filesystem/volume/directory1/directoryx/file         */
/*             /QOPT/VOLN01/DIRA/DIRB/FILE01                     */
```

```
/*  - Complete target path                                              */
/*    Note:  Except for the file the path must already exist.           */
/*    Example: /filesystem/directory1/directoryx/file                   */
/*             /QDLS/DIRA/DIRB/FILE01                                   */
/*                 - or -                                               */
/*           /filesystem/volume/directory1/directoryx/file              */
/*           /QOPT/VOLN01/DIRA/DIRB/FILE01                              */
/*  - Replace existing target file                                      */
/*    *YES - replace existing file                                      */
/*    *NO  - do not replace existing file                               */
/*                                                                      */
/*                                                                      */
/*  LOGIC:                                                              */
/*  - Separate source file length and value                             */
/*  - Ensure source path is converted to upper case                     */
/*  - Separate target file length and value                             */
/*  - Ensure target path is converted to upper case                     */
/*  - Call copy stream file                                             */
/*                                                                      */
/*                                                                      */
/*  EXAMPLE:                                                            */
/*  The example will copy document THISWEEK from folder BILLS           */
/*  to optical volume YEAR1993.  The document will be put into          */
/*  directory /BILLS/DEC as file WEEK50.                                */
/*  Folders are stored in file system DLS (document library services)*/
/*                                                                      */
/*       CPYSTRF SRCFILE('/QDLS/BILLS/THISWEEK')                        */
/*       TGTFILE('/QOPT/YEAR1993/BILLS/DEC/WEEK50')             */
/*       RPLFILE(*NO)                                                   */
/*                                                                      */
/************************************************************************/
  PGM  PARM(&SRCFILE &TGFILE &CPYINFO);

     /*****************************************************************/
     /* Input parameters                                              */
     /*****************************************************************/
     DCL   VAR(&SRCFILE);     TYPE(*CHAR)   LEN(300)
     DCL   VAR(&TGTFILE);     TYPE(*CHAR)   LEN(300)
     DCL   VAR(&CPYINFO);     TYPE(*CHAR)   LEN(6)

     /*****************************************************************/
     /* Program variables                                             */
     /*****************************************************************/
     DCL   VAR(&SRCLEN);      TYPE(*CHAR)   LEN(4)                  +
                                      VALUE(X'00000000')
     DCL   VAR(&TGTLEN);      TYPE(*CHAR)   LEN(4)                  +
                                      VALUE(X'00000000')
     DCL   VAR(&ERRCODE);     TYPE(*CHAR)   LEN(4)                  +
                                      VALUE(X'00000000')
     DCL   VAR(&COUNT);       TYPE(*DEC)    LEN(5 0)
     DCL   VAR(&TBL);         TYPE(*CHAR)   LEN(10)                 +
                                      VALUE('QSYSTRNTBL')
     DCL   VAR(&LIB);         TYPE(*CHAR)   LEN(10)                 +
                                      VALUE('QSYS      ')

     /*****************************************************************/
     /* Monitor for any messages sent to this program                 */
     /*****************************************************************/
     MONMSG     MSGID(CPF0000) EXEC(GOTO CMDLBL(DONE))
     MONMSG     MSGID(OPT0000) EXEC(GOTO CMDLBL(DONE))

     /*****************************************************************/
     /* The HFS API needs to be passed the file and the file length. */
     /* By coding the VARY(*YES) parameter on the command definition */
     /* for the source and target file we are passed the length of   */
     /* entered value as a 2 byte binary field which precedes the    */
     /* actual value entered.                                         */
```

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

     /*************************************************************/
     /* Separate source file length and file value.  Ensure source   */
     /* file is upper case.                                       */
     /*************************************************************/
     CHGVAR VAR(%SST(&SRCLEN 3 2)) VALUE(%SST(&SRCFILE 1 2))
     CHGVAR VAR(%SST(&SRCFILE 1 300)) VALUE(%SST(&SRCFILE 3 298))

     CHGVAR VAR(&COUNT); VALUE(%BIN(&SRCLEN 3 2))
     CALL  QDCXLATE (&COUNT       +
                     &SRCFILE     +
                     &TBL         +
                     &LIB)

     /*************************************************************/
     /* Separate target file length and file value.  Ensure target   */
     /* file is upper case.                                       */
     /*************************************************************/
     CHGVAR VAR(%SST(&TGTLEN 3 2)) VALUE(%SST(&TGTFILE 1 2))
     CHGVAR VAR(%SST(&TGTFILE 1 300)) VALUE(%SST(&TGTFILE 3 298))

     CHGVAR VAR(&COUNT); VALUE(%BIN(&TGTLEN 3 2))
     CALL  QDCXLATE (&COUNT       +
                     &TGTFILE     +
                     &TBL         +
                     &LIB)

     /*************************************************************/
     /* Call the copy stream file HFS API to copy the source file to */
     /* the target file.                                          */
     /*************************************************************/
     CALL  QHFCPYSF (&SRCFILE     +
                     &SRCLEN      +
                     &CPYINFO     +
                     &TGTFILE     +
                     &TGTLEN      +
                     &ERRCODE)

     SNDPGMMSG MSG('CPYSTRF completed successfully')
     RETURN

DONE:
     SNDPGMMSG  MSGID(OPT0125) MSGF(QSYS/QCPFMSG)          +
                MSGDTA(CPYSTRF) MSGTYPE(*ESCAPE)
     RETURN

ENDPGM
```

Copy Database File to Optical: command source

```
/*********************************************************************/
/*                                                                   */
/* COMMAND NAME:    CPYDBOPT                                         */
/*                                                                   */
/* COMMAND TITLE:  Copy Database to Optical                         */
/*                                                                   */
/* DESCRIPTION:     Copy database file to an optical file           */
/*                                                                   */
/*********************************************************************/
CPYDBOPT:   CMD        PROMPT('Copy DB to Optical')

     PARM      KWD(FRMFILE) TYPE(QUAL1) MIN(1)        +
               PROMPT('From file')

     PARM      KWD(FRMMBR) TYPE(*NAME)  LEN(10)       +
               SPCVAL((*FIRST)) EXPR(*YES) MIN(1)    +
```

```
               PROMPT('From member')

     PARM      KWD(TGTFILE) TYPE(*CHAR)  LEN(300)     +
                     MIN(1)  EXPR(*YES)                   +
                      PROMPT('Target file')
QUAL1:     QUAL      TYPE(*NAME) LEN(10)
     QUAL      TYPE(*NAME) LEN(10) DFT(*LIBL)         +
                     SPCVAL((*LIBL) (*CURLIB))            +
                      PROMPT('Library')
```

Copy Database File to Optical: CL program source

This CL example can be used to copy a member from a database file to optical storage.

```
/*********************************************************************/
/*                                                                   */
/*  PROGRAM: CPYDBOPT (Copy Database to Optical)                     */
/*                                                                   */
/*                                                                   */
/*  DESCRIPTION:                                                     */
/*  This is the CL program for sample CL command CPYDBOPT.  This     */
/*  program can be used to copy a member from a database file to     */
/*  optical storage.                                                 */
/*                                                                  */
/*                                                                   */
/*  DEPENDENCIES:                                                    */
/*  - The sample command and program CPYSTRF exists.                 */
/*  - There is an existing folder named OPTICAL.FLR                  */
/*    This folder is used for temporary storage when copying         */
/*    from database to optical.  It is assumed that this folderis    */
/*    empty and that the user will delete anything which gets        */
/*    copied into it.                                                */
/*                                                                   */
/*                                                                   */
/* INPUT PARAMETERS:                                                 */
/*  -From file                                                       */
/*  - From member                                                    */
/*  - Complete target path                                          */
/*     Assumption: - Except for the file the complete path currently  */
/*                   exists.                                         */
/*                 - File does not currently exist.                  */
/*     Example: /filesystem/volume/directory1/directoryx/file        */
/*              /QOPT/VOLN01/DIRA/DIRB/FILE01                        */
/*                                                                   */
/*                                                                   */
/*  LOGIC:                                                           */
/*  - Separate file and library                                      */
/*  - Copy file to folder                                            */
/*  - Build source file                                              */
/*  - Copy file from Document Library Service (DLS) to OPT           */
/*                                                                   */
/*                                                                   */
/*  EXAMPLE:                                                         */
/*  The example will copy member MYMEMBER in file MYFILE in library  */
/*  MYLIB to optical storage. It will be stored as file              */
/*  MYFILE.MYMEMBER in directory /MYLIB on volume VOLN01.            */
/*                                                                   */
/*        CPYDBOPT FRMFILE(MYLIB/MYFILE)                             */
/*                 FRMMBR(MYMEMBER)                                  */
/*                 TGTFILE('/QOPT/VOLN01/MYLIB/MYFILE.MYMEMBER')     */
/*                                                                   */
/*********************************************************************/
  PGM  PARM(&FROMFILE &FROMMBR &TGTFILE);

     /*************************************************************/
     /* Input parameters                                          */
     /*************************************************************/
```

```
     DCL   VAR(&FROMFILE);    TYPE(*CHAR)   LEN(20)
     DCL   VAR(&FROMMBAR);     TYPE(*CHAR)   LEN(10)
     DCL   VAR(&TGTFILE);     TYPE(*CHAR)   LEN(300)

     /*************************************************************/
     /* Program variables                                         */
     /*************************************************************/
     DCL   VAR(&FILE);        TYPE(*CHAR)   LEN(10)
     DCL   VAR(&LIB);         TYPE(*CHAR)   LEN(10)
     DCL   VAR(&SRCFILE);     TYPE(*CHAR)   LEN(28)            +
     VALUE('/QDLS/OPTICAL.FLR/xxxxxxxxxx')

     /*************************************************************/
     /* Monitor for all messages sent to this program             */
     /*************************************************************/
     MONMSG     MSGID(CPF0000) EXEC(GOTO CMDLBL(DONE))
     MONMSG     MSGID(IWS0000) EXEC(GOTO CMDLBL(DONE))
     MONMSG     MSGID(OPT0000) EXEC(GOTO CMDLBL(DONE))

     /*************************************************************/
     /* Separate file and library names then copy the DB file to a   */
     /* PC folder.                                                */
     /*************************************************************/
     CHGVAR VAR(&FILE); VALUE(%SST(&FROMFILE 1 10))
     CHGVAR VAR(&LIB);  VALUE(%SST(&FROMFILE 11 10))

     CPYTOPCD FROMFILE(&LIB/&FILE);                          +
              TOFLR(OPTICAL.FLR)                             +
              FROMMBR(&FROMMBR);                             +
              TRNTBL(*NONE)

     /*************************************************************/
     /* Complete the source file path name with the member and copy  */
     /* the stream file from DLS to optical                       */
     /*************************************************************/
     CHGVAR VAR(%SST(&SRCFILE 19 10)) VALUE(&FROMMBR);

     CPYSTRF  SRCFILE(&SRCFILE);                             +
              TGTFILE(&TGTFILE);

     SNDPGMMSG MSG('CPYDBOPT completed successfully')
     RETURN

DONE:
     SNDPGMMSG  MSGID(OPT0125) MSGF(QSYS/QCPFMSG) +
                MSGDTA(CPYDBOPT) MSGTYPE(*ESCAPE)
     RETURN

ENDPGM
```

Copy Spooled File to Optical: command source

```
/*********************************************************************/
/*                                                                   */
/* COMMAND NAME:   CPYSPLFOPT                                       */
/*                                                                   */
/* COMMAND TITLE: Copy Spooled File to Optical                       */
/*                                                                   */
/* DESCRIPTION:    Copy spooled file to an optical file              */
/*                                                                   */
/*********************************************************************/
CPYSPLFO:    CMD         PROMPT('Copy Spooled File to Optical')

     PARM        KWD(FRMFILE) TYPE(*NAME) LEN(10)      +
                       MIN(1)                               +
                        PROMPT('From file')
```

```
     PARM        KWD(TGTFILE) TYPE(*CHAR)  LEN(300)    +
                        MIN(1)  EXPR(*YES)                 +
                 PROMPT('Target file')

     PARM        KWD(JOB) TYPE(Q2)                       +
                         DFT(*) SNGVAL(*)                    +
                         MIN(0) MAX(1)                       +
                        PROMPT('Jobname')

     PARM        KWD(SPLNBR) TYPE(*CHAR) LEN(5)          +
                        SPCVAL((*ONLY) (*LAST)) DFT(*ONLY)   +
                         PROMPT('Spool number')

Q2:        QUAL        TYPE(*NAME) LEN(10)                    +
                      MIN(1)                                 +
                         EXPR(*YES)

     QUAL        TYPE(*NAME) LEN(10)                     +
                         EXPR(*YES)                           +
                        PROMPT('User')

     QUAL        TYPE(*CHAR) LEN(6)                      +
                         RANGE(000000 999999)                +
                      EXPR(*YES) FULL(*YES)                 +
                         PROMPT('Number')
```

Copy Spooled File to Optical: CL program source

This CL example can be used to copy a spooled file to optical storage.

```
/*********************************************************************/
/*                                                                   */
/*  PROGRAM: CPYSPLFOPT (Copy Spooled File to Optical)               */
/*                                                                   */
/*                                                                   */
/*  DESCRIPTION:                                                     */
/*  This is the CL program for sample CL command CPYSPLFOPT.  This   */
/*  program can be used to copy a spooled file to optical storage.   */
/*                                                                   */
/*                                                                   */
/*  DEPENDENCIES:                                                    */
/*  - The sample command and program CPYDBOPT exists.                */
/*  - The sample command and program CPYSTRF exists.                 */
/*  - There is an existing folder named OPTICAL.FLR                  */
/*    This folder is used for temporary storage when copying         */
/*    from spooled files to optical.  It is assumed that this folder */
/*    is empty and that the user will delete anything which gets     */
/*    copied into it.                                                */
/* - This CL program uses the CL command CPYSPLF to copy the         */
/*   spooled files to a physical file before copying them to         */
/*    optical. When you use the CPYSPLF command to copy              */
/*    a spooled file to a physical file, certain information can     */
/*    be lost or changed. Before using this command please           */
/*    refer to the CL Reference Book for the limitations and         */
/*    restrictions of the CPYSPLF command.                           */
/*  - There is an existing file named LISTINGS in library QUSRSYS.   */
/*    It is assumed that this file contains no existing members      */
/*    and that any members that are created will be deleted by the   */
/*    user.  The record length of the file is 133.                   */
/*                                                                   */
/*                                                                   */
/*  INPUT PARAMETERS:                                                */
/*  - From file                                                      */
/*    Specify the name of the spooled file to be copied.             */
/*  - Target file                                                    */
/*    Assumption: Except for the file the path must already exist.  */
/*    Example: /filesystem/volume/directory1/directoryx/file         */
```

```
/*            /QOPT/VOLN01/DIRA/DIRB/FILE01                          */
/*  - Job                                                            */
/*   Specify the name of the job that created the spooled file       */
/*   which is to be copied.  The possible values are:                */
/*     The job that issued this command is the job that         */
/*     created the spooled file.                                */
/*              - or -                                               */
/*    job-name  Specify the name of the job that created the         */
/*              spooled file.                                        */
/*    user-name  Specify the user name that identifies the user      */
/*               profile under which the job was run.                */
/*    job-number Specify the system assigned job number.             */
/*  - Spool number                                                   */
/*    If there are multiple files for a job specify the files        */
/*    spool number.                                                  */
/*                                                                   */
/*                                                                   */
/*  LOGIC:                                                           */
/*  - Separate job into its three parts: job name, user, job number  */
/*  - Copy spooled files to database                                 */
/*  - Copy database to optical                                       */
/*                                                                   */
/*                                                                   */
/*  EXAMPLE:                                                         */
/*  The example will copy spooled file QSYSPRT spool number 2 which  */
/*  the current process has printed to optical storage.              */
/*  It will be stored on volume YEAR92 in directory                  */
/*  /DEC/WEEK01/MONDAY as file INVOICES                              */
/*                                                                   */
/*       CPYSPLFO SPLFILE(QSYSPRT)                                   */
/*        TGTFILE('/QOPT/YEAR92/DEC/WEEK01/MONDAY/INVOICES')     */
/*        SPLNBR(2)                                                  */
/*                                                                */
/*********************************************************************/
  PGM  PARM(&FROMFILE &TGTFILE &JOB &SPLNBR);

     /*****************************************************************/
     /* Input parameters                                              */
     /*****************************************************************/
     DCL   VAR(&FROMFILE);    TYPE(*CHAR)   LEN(10)
     DCL   VAR(&TGTFILE);     TYPE(*CHAR)   LEN(300)
     DCL   VAR(&JOB);         TYPE(*CHAR)   LEN(26)
     DCL   VAR(&SPLNBR);      TYPE(*CHAR)   LEN(5)

     /*****************************************************************/
     /* Program variables                                             */
     /*****************************************************************/
     DCL   VAR(&JNAME);       TYPE(*CHAR)   LEN(10)
     DCL   VAR(&JUSER);       TYPE(*CHAR)   LEN(10)
     DCL   VAR(&JNUM);        TYPE(*CHAR)   LEN(6)

     /*****************************************************************/
     /* Monitor for all messages that can be signalled                */
     /*****************************************************************/
     MONMSG     MSGID(CPF0000) EXEC(GOTO CMDLBL(DONE))
     MONMSG     MSGID(OPT0000) EXEC(GOTO CMDLBL(DONE))

     /*****************************************************************/
     /* Separate each part of the job name and call the copy spool    */
     /* file command using the current job or the specified name.     */
     /*****************************************************************/
     CHGVAR VAR(&JNAME); VALUE(%SST(&JOB 1 10))
     CHGVAR VAR(&JUSER); VALUE(%SST(&JOB 11 10))
     CHGVAR VAR(&JNUM);  VALUE(%SST(&JOB 21 6))

     IF COND(&JNAME *EQ '*') THEN(DO)
        CPYSPLF  FILE(&FROMFILE);                              +
```

```
                TOFILE(QUSRSYS/LISTINGS)                    +
                TOMBR(&FROMFILE);                           +
                SPLNBR(&SPLNBR);                            +
                CTLCHAR(*FCFC)
        ENDDO

     ELSE DO
        CPYSPLF  FILE(&FROMFILE);                           +
                 TOFILE(QUSRSYS/LISTINGS)                   +
                 TOMBR(&FROMFILE);                          +
                 JOB(&JNUM/&JUSER/&JNAME);                  +
                 SPLNBR(&SPLNBR);                           +
                 CTLCHAR(*FCFC)
        ENDDO

     /*************************************************************/
     /* Copy the database file to optical storage                 */
     /*************************************************************/
     CPYDBOPT FRMFILE(QUSRSYS/LISTINGS)                     +
              FRMMBR(&FROMFILE);                            +
              TGTFILE(&TGTFILE);

     SNDPGMMSG MSG('CPYSPLFOPT completed successfully')
     RETURN

DONE:
     SNDPGMMSG  MSGID(OPT0125) MSGF(QSYS/QCPFMSG)           +
                MSGDTA(CPYSPLFOPT) MSGTYPE(*ESCAPE)

     RETURN

ENDPGM
```

Copy Optical to Database: command source

```
/*********************************************************************/
/*                                                                   */
/* COMMAND NAME:   CPYOPTDB                                          */
/*                                                                   */
/* COMMAND TITLE: Copy Optical to Database                           */
/*                                                                   */
/* DESCRIPTION:    Copy optical file to database file                */
/*                                                                   */
/*********************************************************************/
CPYOPTDB:    CMD       PROMPT('Copy Optical to DB ')

     PARM      KWD(SRCFILE) TYPE(*CHAR)  LEN(300)    +
                       MIN(1)  EXPR(*YES)                  +
                       PROMPT('Source file')

     PARM      KWD(TOFILE) TYPE(QUAL1) MIN(1)        +
                       PROMPT('To file')

     PARM      KWD(TOMBR) TYPE(*NAME)  LEN(10)       +
                      SPCVAL((*FIRST)) EXPR(*YES) MIN(1)    +
                     PROMPT('To member')

QUAL1:      QUAL     TYPE(*NAME) LEN(10)

     QUAL      TYPE(*NAME) LEN(10) DFT(*LIBL)        +
                     SPCVAL((*LIBL) (*CURLIB))             +
                       PROMPT('Library')
```

Copy Optical to Database: CL program source

This CL example can be used to copy a file from an optical volume to a member of an existing file on a database.

```
/*****************************************************************/
/*                                                               */
/*  PROGRAM: CPYOPTDB (Copy Optical to Database)                 */
/*                                                               */
/*                                                              */
/*  DESCRIPTION:                                                 */
/*  This is the CL program for sample CL command CPYOPTDB.  This */
/*  program can be used to copy a file which is on optical       */
/*  storage to a member of an existing file.                     */
/*                                                               */
/*                                                               */
/*  DEPENDENCIES:                                                */
/*  - The sample command and program CPYSTRF exist.              */
/*  - There is an existing folder named OPTICAL.FLR              */
/*    This folder is used for temporary storage when copying     */
/*    from optical to database.  It is assumed that this folder is */
/*    empty and that the user will delete anything which gets    */
/*    copied into it.                                            */
/*                                                               */
/*                                                               */
/* INPUT PARAMETERS:                                             */
/*  - Complete source path                                       */
/*     Example: /filesystem/volume/directory1/directoryx/file    */
/*              /QOPT/VOLN01/DIRA/DIRB/FILE01                    */
/*  - To file                                                    */
/*    Assumptions:                                               */
/*    - Target library already exists.                           */
/*    - Target file already exists and has the same attributes   */
/*      as that which contained the original file.               */
/*  - To member                                                  */
/*                                                               */
/*                                                               */
/*  LOGIC:                                                       */
/* - Build target file                                           */
/*  - Copy file from OPT to Document Library Services (DLS)      */
/*  - Separate file and library                                  */
/*  - Copy from folder to database file                          */
/*                                                               */
/*                                                              */
/*  EXAMPLE:                                                     */
/*  The example will copy file invoices which is in directory    */
/*  DEC on volume YEAR1992.  INVOICES was originally a spooled file */
/*  which had a record length of 133.  It will be placed in file */
/*  LISTINGS which is in library QUSRSYS as member INVOCDEC92.   */
/*                                                               */
/*        CPYDBOPT TGTFILE('/QOPT/YEAR1992/DEC/INVOICES')        */
/*                 TOFILE(QUSRSYS/LISTINGS)                      */
/*                 TOMBR(INVOCDEC92)                             */
/*                                                               */
/*****************************************************************/
  PGM  PARM(&SRCFILE &TOFILE &TOMBR);

     /*************************************************************/
     /* Input parameters                                          */
     /*************************************************************/
     DCL   VAR(&SRCFILE);     TYPE(*CHAR)   LEN(300)
     DCL   VAR(&TOFILE);      TYPE(*CHAR)   LEN(20)
     DCL   VAR(&TOMBR);       TYPE(*CHAR)   LEN(10)

     /*************************************************************/
     /* Program variables                                         */
     /*************************************************************/
```

```
     DCL   VAR(&FILE);        TYPE(*CHAR)   LEN(10)
     DCL   VAR(&LIB);         TYPE(*CHAR)   LEN(10)
     DCL   VAR(&TGTFILE);     TYPE(*CHAR)   LEN(28)           +
                                   VALUE('/QDLS/OPTICAL.FLR/xxxxxxxxxx')

     /*************************************************************/
     /* Monitor for all messages signalled                        */
     /*************************************************************/
     MONMSG     MSGID(CPF0000) EXEC(GOTO CMDLBL(DONE))
     MONMSG     MSGID(IWS0000) EXEC(GOTO CMDLBL(DONE))
     MONMSG     MSGID(OPT0000) EXEC(GOTO CMDLBL(DONE))

     /*************************************************************/
     /* Build the target file name and copy the stream file from  */
     /* optical to DLS                                            */
     /*************************************************************/
     CHGVAR VAR(%SST(&TGTFILE 19 10)) VALUE(&TOMBR);

     CPYSTRF  SRCFILE(&SRCFILE);                          +
              TGTFILE(&TGTFILE);

     /*************************************************************/
     /* Separate the file and library names.  Copy the folder to DB.  */
     /*************************************************************/
     CHGVAR VAR(&FILE); VALUE(%SST(&TOFILE 1 10))
     CHGVAR VAR(&LIB);  VALUE(%SST(&TOFILE 11 10))

     CPYFRMPCD FROMFLR(OPTICAL.FLR)                       +
               TOFILE(&LIB/&FILE);                        +
               FROMDOC(&TOMBR);                           +
               TOMBR(&TOMBR);                             +
               TRNTBL(*NONE)

     SNDPGMMSG MSG('CPYOPTDB completed successfully')
     RETURN

DONE:
     SNDPGMMSG  MSGID(OPT0125) MSGF(QSYS/QCPFMSG)         +
                MSGDTA(CPYOPTDB) MSGTYPE(*ESCAPE)
     RETURN

ENDPGM
```

Related information for Optical device programming

Web sites and other information center topic collections contain information that relates to the Optical device programming topic collection. You can view or print any of the PDF files.

Web sites

- Optical Storage (www-03.ibm.com/servers/eserver/iseries/optical)
- Optical Storage: System i application software (www.ibm.com/servers/eserver/iseries/optical/applications/applications.htm)

Other information

- Integrated file system
- Optical storage
- Application programming interfaces

Related reference

"PDF file for Optical device programming" on page 1
You can view and print a PDF file of this information.

Code license and disclaimer information

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, IBM License Agreement for Machine Code, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This Optical device programming publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i5/OS.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

BladeCenter
i5/OS
IBM
IBM (logo)
Integrated Language Environment
System i
System x

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED ″AS-IS″ AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Printed in USA