



iSeries

# Hierarchical File System APIs

*Version 5 Release 3*







@server

iSeries

Hierarchical File System APIs

*Version 5 Release 3*

**Note**

Before using this information and the product it supports, be sure to read the information in "Notices," on page 121.

**Sixth Edition (August 2005)**

This edition applies to version 5, release 3, modification 0 of Operating System/400 (product number 5722-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.

© Copyright International Business Machines Corporation 1998, 2005. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Hierarchical File System APIs . . . . .</b>	<b>1</b>
APIs . . . . .	2
Hierarchical File System APIs. . . . .	2
Control File System (QHFCFLFS) API . . . . .	4
Required Parameter Group . . . . .	4
Error Messages . . . . .	5
List Registered File Systems (QHFLSTFS) API . . . . .	6
Required Parameter Group . . . . .	6
HFSL0100 Format. . . . .	6
Input Parameter Section . . . . .	7
List Data Section . . . . .	7
Field Descriptions . . . . .	7
Error Messages . . . . .	7
Create Directory (QHFCRTDR) API. . . . .	8
Required Parameter Group . . . . .	8
Error Messages . . . . .	9
Delete Directory (QHFDLTDR) API . . . . .	10
Required Parameter Group . . . . .	10
Error Messages . . . . .	10
Rename Directory (QHFRNMDR) API . . . . .	11
Required Parameter Group . . . . .	12
Error Messages . . . . .	12
Change File Pointer (QHFCGFP) API . . . . .	13
Required Parameter Group . . . . .	13
How to Move the File Pointer . . . . .	14
Examples . . . . .	15
Error Messages . . . . .	15
Close Stream File (QHFCLOSF) API . . . . .	16
Required Parameter Group . . . . .	16
Error Messages . . . . .	16
Force Buffered Data (QHFFRCSF) API . . . . .	17
Required Parameter Group . . . . .	17
Error Messages . . . . .	17
Get Stream File Size (QHFGTFSZ) API . . . . .	18
Required Parameter Group . . . . .	18
Error Messages . . . . .	19
Lock and Unlock Range in Stream File (QHFLULSF) API . . . . .	19
Required Parameter Group . . . . .	20
Error Messages . . . . .	21
Open Stream File (QHFOFNSF) API . . . . .	22
Required Parameter Group . . . . .	22
Lock and Access Modes . . . . .	25
Lock Modes . . . . .	25
Access Modes . . . . .	25
Error Messages . . . . .	26
Read from Stream File (QHFRDSF) API . . . . .	27
Required Parameter Group . . . . .	28
Error Messages . . . . .	28
Set Stream File Size (QHSETSFSZ) API . . . . .	29
Required Parameter Group . . . . .	29
Error Messages . . . . .	30
Write to Stream File (QHFWRFSF) API . . . . .	30
Required Parameter Group . . . . .	31
Error Messages . . . . .	31
Copy Stream File (QHFCPYFSF) API . . . . .	32
Required Parameter Group . . . . .	32
Error Messages . . . . .	33
Delete Stream File (QHFDLTSF) API . . . . .	35
Required Parameter Group . . . . .	35
Error Messages . . . . .	35
Move Stream File (QHFMVFSF) API . . . . .	36
Required Parameter Group . . . . .	37
Error Messages . . . . .	37
Rename Stream File (QHFRNMSF) API . . . . .	38
Required Parameter Group . . . . .	38
Error Messages . . . . .	39
Change Directory Entry Attributes (QHFCGAT) API . . . . .	40
Required Parameter Group . . . . .	40
Error Messages . . . . .	41
Close Directory (QHFCLODR) API . . . . .	42
Required Parameter Group . . . . .	42
Error Messages . . . . .	42
Open Directory (QHFOFNDR) API . . . . .	43
Required Parameter Group . . . . .	43
Error Messages . . . . .	45
Read Directory Entries (QHFRDDR) API . . . . .	46
Required Parameter Group . . . . .	46
Data Buffer . . . . .	47
Error Messages . . . . .	48
Retrieve Directory Entry Attributes (QHFRTVAT) API . . . . .	49
Required Parameter Group . . . . .	49
Error Messages . . . . .	50
File System Registration APIs . . . . .	51
Deregister File System (QHFDREGFS) API . . . . .	52
Required Parameter Group . . . . .	53
Error Messages . . . . .	53
Register File System (QHFRGFS) API. . . . .	54
Authorities and Locks . . . . .	54
Required Parameter Group . . . . .	54
User Authorizations and Locks for File System Functions . . . . .	57
Error Messages . . . . .	57
Exit Programs . . . . .	58
Exit Program for Change Directory Entry Attributes (QHFCGAT) API . . . . .	58
Required Parameter Group . . . . .	58
API Functions . . . . .	59
Exit Program Requirements . . . . .	59
Error Messages for Exit Program Use. . . . .	59
Exit Program for Change File Pointer (QHFCGFP) API . . . . .	60
Required Parameter Group . . . . .	61
API Functions . . . . .	61
Exit Program Requirements . . . . .	61
Error Messages for Exit Program Use. . . . .	61
Exit Program for Close Directory (QHFCLODR) API . . . . .	62
Required Parameter Group . . . . .	62
Optional Parameter Group . . . . .	62
API Functions . . . . .	63

Exit Program Requirements . . . . .	63	Required Parameter Group . . . . .	85
Error Messages for Exit Program Use. . . . .	63	API Functions . . . . .	86
Exit Program for Close Stream File (QHFCLFSF)		Exit Program Requirements . . . . .	86
API . . . . .	64	Error Messages for Exit Program Use. . . . .	86
Required Parameter Group . . . . .	64	Exit Program for Open Directory (QHFOPNDR) API	88
Optional Parameter Group . . . . .	64	Required Parameter Group . . . . .	88
API Functions . . . . .	65	API Functions . . . . .	89
Exit Program Requirements . . . . .	65	Exit Program Requirements . . . . .	89
Error Messages for Exit Program Use. . . . .	65	Error Messages for Exit Program Use. . . . .	89
Exit Program for Control File System (QHFCTLFS)		Exit Program for Open Stream File (QHFOPNSF)	
API . . . . .	66	API . . . . .	90
Required Parameter Group . . . . .	66	Required Parameter Group . . . . .	91
API Functions . . . . .	67	API Functions . . . . .	91
Exit Program Requirements . . . . .	67	Exit Program Requirements . . . . .	92
Error Messages for Exit Program Use. . . . .	67	Error Messages for Exit Program Use. . . . .	92
Exit Program for Copy Stream File (QHFCPYSF)		Exit Program for Read Directory Entries	
API . . . . .	68	(QHFRDDR) API . . . . .	93
Required Parameter Group . . . . .	68	Required Parameter Group . . . . .	93
API Functions . . . . .	69	API Functions . . . . .	94
Exit Program Requirements . . . . .	72	Exit Program Requirements . . . . .	94
Error Messages for Exit Program Use. . . . .	72	Error Messages for Exit Program Use. . . . .	94
Exit Program for Create Directory (QHFCRTDR)		Exit Program for Read from Stream File (QHFRDSF)	
API . . . . .	73	API . . . . .	95
Required Parameter Group . . . . .	74	Required Parameter Group . . . . .	95
API Functions . . . . .	74	API Functions . . . . .	96
Exit Program Requirements . . . . .	74	Exit Program Requirements . . . . .	96
Error Messages for Exit Program Use. . . . .	74	Error Messages for Exit Program Use. . . . .	96
Exit Program for Delete Directory (QHFDLDR) API	75	Exit Program for Rename Directory (QHFRNMDR)	
Required Parameter Group . . . . .	76	API . . . . .	97
API Functions . . . . .	76	Required Parameter Group . . . . .	97
Exit Program Requirements . . . . .	76	API Functions . . . . .	98
Error Messages for Exit Program Use. . . . .	76	Exit Program Requirements . . . . .	98
Exit Program for Delete Stream File (QHFDLTSF)		Error Messages for Exit Program Use. . . . .	98
API . . . . .	77	Exit Program for Rename Stream File (QHFRNMSF)	
Required Parameter Group . . . . .	77	API . . . . .	99
API Functions . . . . .	78	Required Parameter Group . . . . .	99
Exit Program Requirements . . . . .	78	API Functions . . . . .	100
Error Messages for Exit Program Use. . . . .	78	Exit Program Requirements . . . . .	100
End Job Session Exit Program . . . . .	79	Error Messages for Exit Program Use . . . . .	100
Required Parameter Group . . . . .	79	Exit Program for Retrieve Directory Entry	
Error Messages for Exit Program Use. . . . .	79	Attributes (QHFRTVAT) API . . . . .	101
Exit Program for Force Buffered Data (QHFFRCSF)		Required Parameter Group . . . . .	101
API . . . . .	79	API Functions . . . . .	102
Required Parameter Group . . . . .	80	Exit Program Requirements . . . . .	102
API Functions . . . . .	80	Error Messages for Exit Program Use . . . . .	102
Exit Program Requirements . . . . .	80	Exit Program for Set Stream File Size (QHFSETS)	
Error Messages for Exit Program Use. . . . .	80	API. . . . .	103
Exit Program for Get Stream File Size (QHFGETSZ)		Required Parameter Group . . . . .	103
API . . . . .	81	API Functions . . . . .	104
Required Parameter Group . . . . .	81	Exit Program Requirements . . . . .	104
API Functions . . . . .	82	Error Messages for Exit Program Use . . . . .	104
Exit Program Requirements . . . . .	82	Start Job Session Exit Program. . . . .	105
Error Messages for Exit Program Use. . . . .	82	Required Parameter Group . . . . .	105
Exit Program for Lock and Unlock Range in Stream		Error Messages for Exit Program Use . . . . .	106
File (QHFLULSF) API . . . . .	83	Exit Program for Write to Stream File	
Required Parameter Group . . . . .	83	(QHFWRTSF) API . . . . .	106
API Functions . . . . .	84	Required Parameter Group . . . . .	106
Exit Program Requirements . . . . .	84	API Functions . . . . .	107
Error Messages for Exit Program Use. . . . .	84	Exit Program Requirements . . . . .	107
Exit Program for Move Stream File (QHFMVFSF)		Error Messages for Exit Program Use . . . . .	107
API . . . . .	85	Concepts . . . . .	108

HFS Concepts . . . . .	108
HFS Use—Requirements. . . . .	109
HFS Directory Entry Attributes . . . . .	109
Standard HFS Directory Entry Attributes . . . . .	109
Other HFS Directory Entry Attributes . . . . .	111
HFS Attribute Information Table . . . . .	112
HFS Attribute Selection Table . . . . .	113
New File Systems . . . . .	114
Scenario: HFS Exit Program . . . . .	114
Creating a Directory in Your File System . . . . .	115
Enabling Your File System to HFS . . . . .	115
How HFS Support Processes a File System Job . . . . .	116
First Call to File System . . . . .	116
Subsequent Calls to File System . . . . .	117

End of Job . . . . .	117
Scenario: File System Job . . . . .	117
File System—Error Messages . . . . .	117
Standard HFS API and Exit Program Functions . . . . .	117
Standard HFS API Functions . . . . .	117
Standard HFS Exit Program Requirements. . . . .	118

<b>Appendix. Notices . . . . .</b>	<b>121</b>
Trademarks . . . . .	122
Terms and conditions for downloading and printing publications . . . . .	123
Code disclaimer information . . . . .	124





---

## Hierarchical File System APIs

For information on the Hierarchical File System (HFS) APIs, see the following:

- “HFS Concepts” on page 108
- “HFS Use—Requirements” on page 109
- “HFS Directory Entry Attributes” on page 109

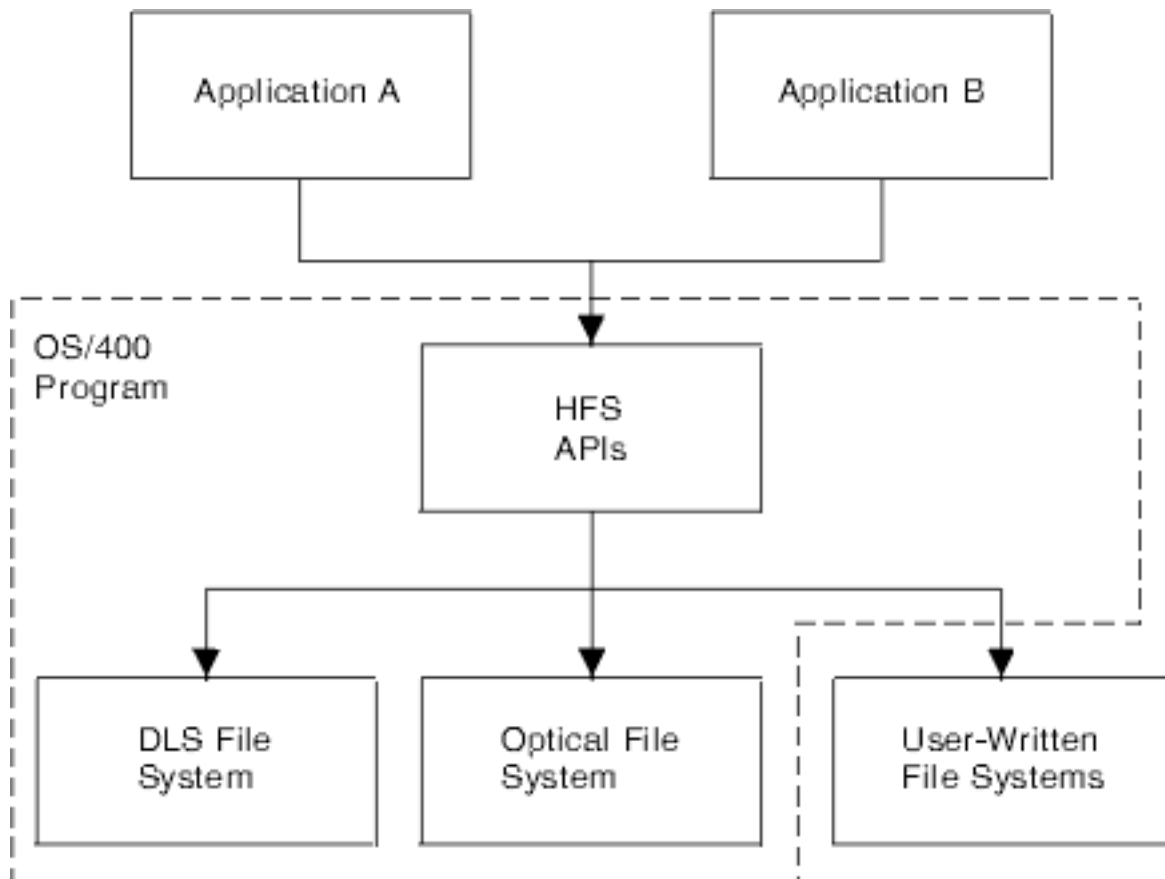
The hierarchical file system (HFS) APIs include:


- “Hierarchical File System APIs” on page 2
- “File System Registration APIs” on page 51

The HFS APIs and the functions they support are part of the OS/400<sup>(R)</sup> program. These APIs provide applications with a single, consistent interface to the file systems registered with the hierarchical file system on your iSeries server. They automatically support the document library services (DLS) file system and the optical file system (QOPT), and they can also support user-written file systems.

With HFS APIs you can work with nonrelational data stored in objects, such as directories and files in existing file systems. You can also perform such tasks as creating and deleting directories and files, reading from and writing to files, and changing the directory entry attributes of files and directories.

The following diagram shows the relationship of the HFS APIs to your applications and to other file systems:



If you are working with the optical file system (QOPT), refer to the Optical Support book . If you are not writing applications, but instead are creating or installing a new file system for other programmers to use with the HFS APIs, turn to “File System Registration APIs” on page 51.

Top | APIs by category

---

## APIs

These are the APIs for this category.

---

### Hierarchical File System APIs

The HFS APIs allow you to work with files and directories and with the data stored in files. These APIs perform a wide variety of tasks in the following categories:

- **File system management APIs** help you manage your use of file systems in general. The file system management APIs are:
  - “Control File System (QHFCTLFS) API” on page 4 (QHFCTLFS) allows your applications to issue file-system-specific commands.
  - “List Registered File Systems (QHFLSTFS) API” on page 6 (QHFLSTFS) lists the file systems that are registered on your system and thus available for use through the HFS APIs.
- **Directory management APIs** allow you to perform general maintenance tasks for directories in hierarchical file systems. The directory management APIs are:
  - “Create Directory (QHFCRTDR) API” on page 8 (QHFCRTDR) creates a new directory and its directory entry. Except for the directory being created, all directories in the path must exist.
  - “Delete Directory (QHFDLTDR) API” on page 10 (QHFDLTDR) deletes a single, empty directory.
  - “Rename Directory (QHFRNMDR) API” on page 11 (QHFRNMDR) renames a single directory.
- **File input and output APIs** allow you to work with the contents of files in hierarchical file systems. The APIs work with **stream files**, which are files that have varying lengths and no conventional record structure. Stream files are also called byte-stream files, or simply files. The file input and output APIs are:
  - “Change File Pointer (QHFCHGFP) API” on page 13 (QHFCHGFP) allows you to change the location of the current read/write position in the file.
  - “Close Stream File (QHFCLOSF) API” on page 16 (QHFCLOSF) closes the specified stream file, releasing any locks on the file or ranges within the file.
  - “Force Buffered Data (QHFFRCSF) API” on page 17 (QHFFRCSF) forces data from a buffer into nonvolatile storage. (**Nonvolatile storage** is any storage area whose contents are not lost when power is cut off or when the system is loaded.)
  - “Get Stream File Size (QHFGETSZ) API” on page 18 (QHFGETSZ) returns the current size of a stream file’s data, in bytes, as of the last write operation to the file.
  - “Lock and Unlock Range in Stream File (QHFLULSF) API” on page 19 (QHFLULSF) allows you to lock and unlock parts of files.
  - “Open Stream File (QHFOPNSF) API” on page 22 (QHFOPNSF) opens and optionally creates a single stream file.
  - “Read from Stream File (QHFRDSF) API” on page 27 (QHFRDSF) reads a specified number of bytes from a stream file opened with an access mode of read only or read/write.
  - “Set Stream File Size (QHFSETSZ) API” on page 29 (QHFSETSZ) sets the size of a stream file in bytes.
  - “Write to Stream File (QHFWRTSF) API” on page 30 (QHFWRTSF) writes bytes to a stream file.
- **File management APIs** allow you to perform general maintenance tasks for files in hierarchical file systems. The file management APIs are:

- “Copy Stream File (QHFCPYSF) API” on page 32 (QHFCPYSF) copies an existing stream file into another stream file and optionally renames the copy.
- “Delete Stream File (QHFDLTSF) API” on page 35 (QHFDLTSF) deletes a single stream file. Both the directory entry associated with the file and all data contained in the file object are deleted.
- “Move Stream File (QHFMVVSF) API” on page 36 (QHFMVVSF) moves a single stream file from one directory to another and optionally changes the file’s name. The file’s attributes are not changed.
- “Rename Stream File (QHFRNMSF) API” on page 38 (QHFRNMSF) renames a stream file in the same path.
- **Directory entry information APIs** allow you to work with the directory entries for files and directories in hierarchical file systems. The directory entry information APIs are:
  - “Change Directory Entry Attributes (QHFCGAT) API” on page 40 (QHFCGAT) changes the attributes of a specified directory entry for an existing file or directory.
  - “Close Directory (QHFCLODR) API” on page 42 (QHFCLODR) closes a specified directory that was opened using the Open Directory (QHFOPNDR) API.
  - “Open Directory (QHFOPNDR) API” on page 43 (QHFOPNDR) opens the specified directory so its directory entries can be read.
  - “Read Directory Entries (QHFRDDR) API” on page 46 (QHFRDDR) reads one or more directory entries from a directory opened with the Open Directory (QHFOPNDR) API.
  - “Retrieve Directory Entry Attributes (QHFRTVAT) API” on page 49 (QHFRTVAT) retrieves attribute information from a specified directory entry for a directory or file.

Top | “Hierarchical File System APIs,” on page 1 | APIs by category

---

## Control File System (QHFCTLFS) API

Required Parameter Group:	
1	Open file handle
<b>Input</b>	Char(16)
2	File system name
<b>Input</b>	Char(10)
3	Input data buffer
<b>Input</b>	Char(*)
4	Input data buffer length
<b>Input</b>	Binary(4)
5	Output data buffer
<b>Output</b>	Char(*)
6	Output data buffer length
<b>Input</b>	Binary(4)
7	Length of data returned
<b>Output</b>	Binary(4)
8	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Control File System (QHFCTLFS) API transmits commands to your file system to be performed. The commands must be defined by the file system.

### Required Parameter Group

#### Open file handle

INPUT; CHAR(16)

The identifier returned when the file was opened with the Open Stream File (QHFOPNSF) API. If you specify a file system in the file system name parameter, this parameter is ignored.

#### File system name

INPUT; CHAR(10)

The registered file system to send the request to. Valid values are:

*name*

The name of the registered file system. The open file handle parameter is ignored.

*\*HANDLE*

A special value indicating that the registered file system name is derived from the file handle provided in the open file handle parameter. Using the handle provides better performance than using the file system name.

#### Input data buffer

INPUT; CHAR(\*)

The command string to send to the file system. This string differs from file system to file system. The QHFCTLFS API simply passes it on.

**Input data buffer length**

INPUT; BINARY(4)

The length of the input data buffer, in bytes.

**Output data buffer**

OUTPUT; CHAR(\*)

The data returned from the file system.

**Output data buffer length**

INPUT; BINARY(4)

The length of the output data buffer, in bytes.

**Length of data returned**

OUTPUT; BINARY(4)

The length of the data returned by the file system in the output data buffer, in bytes.

**Error code**

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

## Error Messages

Message ID	Error Message Text
CPF1F05 E	Directory handle not valid.
CPF1F25 E	File handle not valid.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F47 E	Buffer overflow occurred.
CPF1F52 E	Error code not valid.
CPF1F53 E	Value for length of data buffer not valid.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F81 E	API specific error occurred.
CPF1F82 E	Function not supported.
CPF1F83 E	File system name &1 not found.
CPF1F85 E	Not authorized to file system &1.
CPF1F87 E	Missing or damaged exit program &2.
CPF1F97 E	File system &1 in use.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## List Registered File Systems (QHFLSTFS) API

Required Parameter Group:

1 Qualified user space name

**Input** Char(20)

2 Format name

**Input** Char(8)

3 Error code

**I/O** Char(\*)

Default Public Authority: \*USE

Threadsafe: No

The List Registered File Systems (QHFLSTFS) API returns information similar to the Display Hierarchical File Systems (DSPHFS) command. The QHFLSTFS API retrieves a list of all file systems that are currently registered and thus available for use through the HFS APIs. The list gives the name, version level, and text description for each file system.

### Required Parameter Group

#### Qualified user space name

INPUT; CHAR(20)

The name of the \*USRSPC object that is to receive the generated list. The first 10 characters give the user space name, and the second 10 characters give the name of the library in which the user space resides. You can use these special values for the library name:

\*CURLIB                    The job's current library

\*LIBL                      The library list

#### Format name

INPUT; CHAR(8)

The format of the information returned. You must use this format name:

HFSL0100                File system list. For details, see "HFSL0100 Format."

#### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

### HFSL0100 Format

The HFSL0100 file system list consists of:

- A user area
- A generic header
- An input parameter section
- A list data section

The user area and generic header are described in the User Space Format for List APIs. When you retrieve list entry information from a user space and you want to increase pointer values or add to variables used in the QUSRTVUS API, you must use the entry size returned in the generic header. The entries can be padded at the end.

The input parameter and list data sections are specific to the HFSL0100 format. They are described below. For a detailed description of each item, see "Field Descriptions."

## Input Parameter Section

Offset		Type	Information
Dec	Hex		
0	0	CHAR(10)	User space name
10	A	CHAR(10)	User space library name
20	14	CHAR(8)	Format name

## List Data Section

Offset		Type	Information
Dec	Hex		
0	0	CHAR(10)	File system name
10	A	CHAR(6)	Version
16	10	CHAR(50)	Text description

## Field Descriptions

**File system name.** The name of the file system when it was registered.

**Format name.** The format of the returned output.

**Text description.** The description of the file system specified at registration time.

**User space library name.** The name of the library containing the user space.

**User space name.** The name of the user space that receives the list.

**Version.** The version the file system supports. This is in the form VxRxMx, where x represents the version, release, and modification levels, respectively.

## Error Messages

Message ID	Error Message Text
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F52 E	Error code not valid.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F81 E	API specific error occurred.
CPF3C21 E	Format name &1 is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF8100 E	All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9801 E	Object &2 in library &3 not found.

**Message ID Error Message Text**

CPF9802 E Not authorized to object &2 in &3.  
 CPF9803 E Cannot allocate object &2 in library &3.  
 CPF9807 E One or more libraries in library list deleted.  
 CPF9808 E Cannot allocate one or more libraries on library list.  
 CPF9810 E Library &1 not found.  
 CPF9820 E Not authorized to use library &1.  
 CPF9830 E Cannot assign library &1.  
 CPF9838 E User profile storage limit exceeded.  
 CPF9872 E Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## Create Directory (QHFCRTDR) API

Required Parameter Group:

<b>1</b>	Path name
<b>Input</b>	Char(*)
<b>2</b>	Length of path name
<b>Input</b>	Binary(4)
<b>3</b>	Attribute information table
<b>Input</b>	Char(*)
<b>4</b>	Length of attribute information table
<b>Input</b>	Binary(4)
<b>5</b>	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Create Directory (QHFCRTDR) API creates a new directory and its directory entry. Except for the directory being created, all directories in the path must exist.

### Required Parameter Group

#### Path name

INPUT; CHAR(\*)

The path name for the new directory. The last element of the path name specifies the directory being created. For example, specifying /QDLS/A/B creates new directory B and adds a directory entry for B to directory A in file system QDLS.

#### Length of path name

INPUT; BINARY(4)

The length of the path name, in bytes.



### Attribute information table

INPUT; CHAR(\*)

The table specifying the attributes for the new directory. The file system determines which standard and extended attributes you can specify. For detailed descriptions of the standard attributes and the format of the table, see “HFS Directory Entry Attributes” on page 109.

If no attributes are specified, the file system’s defaults are used for required information.

### Length of attribute information table

INPUT; BINARY(4)

The length of the attribute information table, in bytes. Valid values are:

*length*  
0

Use the attributes specified in the table.

Use the system defaults for standard attributes instead of using the attributes in the table.

### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

## Error Messages

Message ID	Error Message Text
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F04 E	Directory name already exists.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F09 E	Use of reserved directory name not allowed.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F42 E	Attribute information table not valid.
CPF1F43 E	Attribute name not valid.
CPF1F44 E	Attribute value is not valid.
CPF1F46 E	Use of reserved attribute name not allowed.
CPF1F48 E	Path name not valid.
CPF1F52 E	Error code not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F81 E	API specific error occurred.
CPF1F82 E	Function not supported.
CPF1F83 E	File system name &1 not found.
CPF1F85 E	Not authorized to file system &1.
CPF1F87 E	Missing or damaged exit program &2.
CPF1F97 E	File system &1 in use.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

---

## Delete Directory (QHFDLTDR) API

Required Parameter Group:	
1	Path name
<b>Input</b>	Char(*)
2	Length of path name
<b>Input</b>	Binary(4)
3	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Delete Directory (QHFDLTDR) API deletes a single, empty directory. It cannot contain any directory entries. If this job or another job has already opened the directory with a deny none or deny write lock, it cannot be deleted.

### Required Parameter Group

#### Path name

INPUT; CHAR(\*)

The path name of the directory being deleted. The last element of the path name must be a directory name. You cannot use a one-element path name specifying only the file system.

#### Path name length

INPUT; BINARY(4)

The length of the path name, in bytes.

#### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

### Error Messages

Message ID	Error Message Text
CPF1F0A E	Delete directory operation not allowed.
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F48 E	Path name not valid.
CPF1F52 E	Error code not valid.

**Message ID    Error Message Text**

CPF1F61 E    No free space available on media.  
 CPF1F62 E    Requested function failed.  
 CPF1F63 E    Media is write protected.  
 CPF1F66 E    Storage needed exceeds maximum limit for user profile &1.  
 CPF1F71 E    Exception specific to file system occurred.  
 CPF1F72 E    Internal file system error occurred.  
 CPF1F73 E    Not authorized to use command.  
 CPF1F74 E    Not authorized to object.  
 CPF1F75 E    Error occurred during start-job-session function.  
 CPF1F81 E    API specific error occurred.  
 CPF1F82 E    Function not supported.  
 CPF1F83 E    File system name &1 not found.  
 CPF1F85 E    Not authorized to file system &1.  
 CPF1F87 E    Missing or damaged exit program &2.  
 CPF1F97 E    File system &1 in use.  
 CPF3C90 E    Literal value cannot be changed.  
 CPF9872 E    Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## Rename Directory (QHFRNMDR) API

Required Parameter Group:	
1	Path name
<b>Input</b>	Char(*)
2	Path name length
<b>Input</b>	Binary(4)
3	New directory name
<b>Input</b>	Char(*)
4	New directory name length
<b>Input</b>	Binary(4)
5	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Rename Directory (QHFRNMDR) API renames a single directory. You cannot rename the directory if another job has already opened it. (When a job opens a directory with the Open Directory (QHFOPNDR) API, it specifies a lock mode. The lock mode specifies what other jobs are allowed to do to the directory while the first job has it open.)

## Required Parameter Group

### Path name

INPUT; CHAR(\*)

The path name of the directory being renamed. The last element of the path name must be a directory name. You cannot use a one-element path name specifying only the file system.

### Path name length

INPUT; BINARY(4)

The length of the path name, in bytes.

### New directory name

INPUT; CHAR(\*)

The new name for the directory. Specify only the directory name. Do not specify the path name.

### New directory name length

INPUT; BINARY(4)

The length of the new directory name, in bytes.

### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

## Error Messages

Message ID	Error Message Text
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F03 E	New directory name same as old directory name.
CPF1F04 E	Directory name already exists.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F09 E	Use of reserved directory name not allowed.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F48 E	Path name not valid.
CPF1F52 E	Error code not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F81 E	API specific error occurred.
CPF1F82 E	Function not supported.
CPF1F83 E	File system name &1 not found.
CPF1F85 E	Not authorized to file system &1.
CPF1F87 E	Missing or damaged exit program &2.
CPF1F97 E	File system &1 in use.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

---

## Change File Pointer (QHFCHGFP) API

Required Parameter Group:	
1	Open file handle
<b>Input</b>	Char(16)
2	Move information
<b>Input</b>	Char(6)
3	Distance to move
<b>Input</b>	Binary(4)
4	New offset
<b>Output</b>	Binary(4) Unsigned
5	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Change File Pointer (QHFCHGFP) API moves the file pointer a specified number of bytes forward or backward. (The **file pointer** is the current read/write position in the file.) This file pointer is used by the Read from Stream File (QHFRDSF) and Write to Stream File (QHFWRFSF) APIs. You can also use the QHFCHGFP API to determine the size of a file by moving the pointer to the end of the file.

### Required Parameter Group

#### Open file handle

INPUT; CHAR(16)

The handle returned when the file was opened with the Open Stream File (QHFOPNSF) API.

#### Move information

INPUT; CHAR(6)

Additional information specifying the action to take. The 6 characters of this parameter are:

1

The pointer's starting location. The pointer is moved the distance you specify from this place. For example, specifying 0 here and 5 in the distance to move parameter moves the pointer to a new position 5 bytes from the start of the file. Valid values for the starting location are:

0 The beginning of the file.

1 The pointer's current location.

2 The end of the file. If the distance to move is zero, the new offset parameter returns the file's size.

2-6

Reserved. These characters must be set to blanks.

### Distance to move

INPUT; BINARY(4)

The distance to move the pointer from the starting location, in bytes. A negative value parameter moves the pointer backward in the file. A positive value moves it forward.

The file pointer can be set to any location that can be supported by a 4-byte unsigned value. An error is returned only if the application tries to move the pointer to a negative position or an offset larger than the maximum value that can be stored in a 4-byte unsigned binary number.

Setting the file pointer beyond the end of the file is allowed, but it does not change the file's size. To change the file's size, see "Set Stream File Size (QHFSETS) API" on page 29 or "Write to Stream File (QHFWRTSF) API" on page 30.

For brief examples of how the move information and the distance to move work together, see "How to Move the File Pointer."

### New offset

OUTPUT; BINARY(4) UNSIGNED

The new position of the pointer.

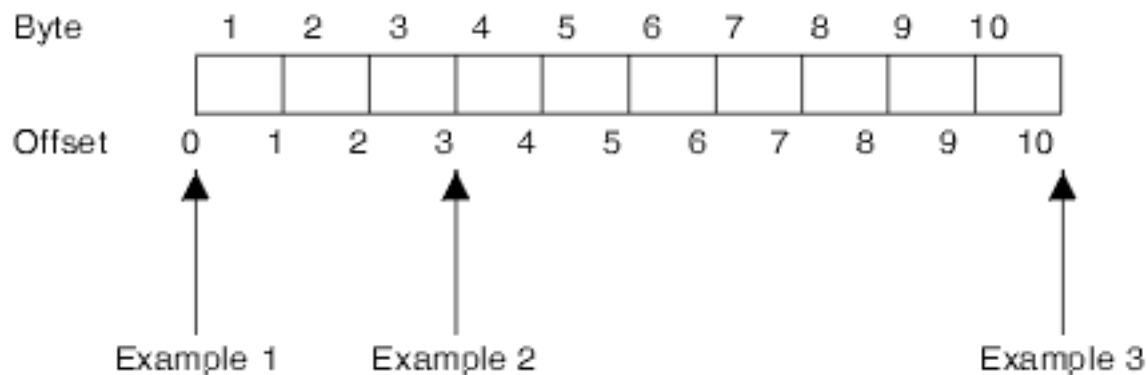
### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error code parameter.

## How to Move the File Pointer

The file pointer represents a position or offset within a file where the next read or write is to take place. It does not actually point to a byte in the file; rather, it points to the gap between bytes. The diagram below represents a 10-byte file as a series of boxes. Each box represents a byte of data in the file.



## Examples

1. To move the file pointer to the beginning of the file, set the start location in the move information parameter to zero (the beginning of the file). Set the distance to move to zero, too.
2. To move the file pointer to offset 3 (that is, pointing to the gap between bytes 3 and 4), set the start location in the move information parameter to zero (the beginning of the file). Set the distance to move to 3.
3. To move the file pointer to the end of the file, set the start location in the move information parameter to 2 (the end of the file). Set the distance to move to zero. The new offset parameter returns the file's size.

## Error Messages

Message ID	Error Message Text
CPF1F2D E	File pointer position not valid.
CPF1F2E E	Range of bytes in file in use.
CPF1F25 E	File handle not valid.
CPF1F28 E	Damaged file.
CPF1F4E E	Move information value not valid.
CPF1F4F E	Distance to move value not valid.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F52 E	Error code not valid.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F82 E	Function not supported.
CPF1F87 E	Missing or damaged exit program &2.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

Top | "Hierarchical File System APIs," on page 1 | APIs by category

---

## Close Stream File (QHFCLOSF) API

Required Parameter Group:

1 Open file handle

**Input** Char(16)

2 Error code

**I/O** Char(\*)

Default Public Authority: \*USE

Threadsafe: No

The Close Stream File (QHFCLOSF) API closes the specified stream file, releasing any locks on the file or ranges within the file. Any data and directory information waiting to be written is forced to nonvolatile storage. Nonvolatile storage is any storage area whose contents are not lost when power is cut off or when the system is loaded.

### Required Parameter Group

#### Open file handle

INPUT; Input Char(16)

The handle returned when the file being closed was opened with the Open Stream File (QHFOFNSF) API.

#### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

### Error Messages

Message ID	Error Message Text
CPF1F06 E	Directory in use.
CPF1F25 E	File handle not valid.
CPF1F28 E	Damaged file.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F52 E	Error code not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F82 E	Function not supported.
CPF1F87 E	Missing or damaged exit program &2.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1



## Force Buffered Data (QHFFRCSF) API

Required Parameter Group:	
1	Files to force
<b>Input</b>	Char(16)
2	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Force Buffered Data (QHFFRCSF) API synchronously forces buffered data and directory entry information out of main storage and into nonvolatile storage for either a specific file or all files opened by a job. (Nonvolatile storage is any storage area whose contents are not lost when power is cut off or when the system is loaded.) Forcing buffered data is similar to closing a file because the data is forced. However, after a force operation, the file remains open, and forcing has no effect on locks and read/write positions.

### Required Parameter Group

#### Files to force

INPUT; CHAR(16)

The files whose data is being forced. Valid values are:

*Open file handle*

Forces the single file that was assigned this handle when opened with the Open Stream File (QHFOPNSF) API.

*Hexadecimal zeros*

Forces all files opened by the job.

#### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

### Error Messages

#### Message ID    Error Message Text

CPF1F2B E	Write operation not allowed to file opened for read only.
CPF1F2E E	Range of bytes in file in use.
CPF1F25 E	File handle not valid.
CPF1F28 E	Damaged file.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F52 E	Error code not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.

**Message ID    Error Message Text**

- CPF1F71 E    Exception specific to file system occurred.
- CPF1F72 E    Internal file system error occurred.
- CPF1F73 E    Not authorized to use command.
- CPF1F74 E    Not authorized to object.
- CPF1F82 E    Function not supported.
- CPF1F86 E    Force all files did not complete successfully.
- CPF1F87 E    Missing or damaged exit program &2.
- CPF3C90 E    Literal value cannot be changed.
- CPF9872 E    Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

## Get Stream File Size (QHFGGETSZ) API

Required Parameter Group:	
1	Open file handle
<b>Input</b>	Char(16)
2	File size
<b>Output</b>	Binary(4) Unsigned
3	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Get Stream File Size (QHFGGETSZ) API returns the current size of a stream file’s data, in bytes, as of the last write operation to the file.

### Required Parameter Group

#### Open file handle

INPUT; CHAR(16)

The file handle returned when the file was opened with the Open Stream File (QHFOPNSF) API.

#### File size

OUTPUT; BINARY(4) UNSIGNED

The number of bytes of data in the file as of the last write operation. This number is either the last offset written plus 1 or the size set with the Set Stream File Size (QHFSETSZ) API whichever is higher.

This size can differ from the value of the QFILSIZE attribute, which is the size of the file as of the last close operation. In addition, it is the size of the file’s data only and does not include the size of any object information stored by the file system. If your file system stores object or other information with the file, the file’s allocated size is larger than this size.

#### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

## Error Messages

Message ID	Error Message Text
CPF1F25 E	File handle not valid.
CPF1F28 E	Damaged file.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F52 E	Error code not valid.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F82 E	Function not supported.
CPF1F87 E	Missing or damaged exit program &2.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## Lock and Unlock Range in Stream File (QHFLULSF) API

Required Parameter Group:	
1	Open file handle
<b>Input</b>	Char(16)
2	Lock information
<b>Input</b>	Char(6)
3	File offset where lock begins
<b>Input</b>	Binary(4) Unsigned
4	Bytes to lock
<b>Input</b>	Binary(4) Unsigned
5	File offset where unlock begins
<b>Input</b>	Binary(4) Unsigned
6	Bytes to unlock
<b>Input</b>	Binary(4) Unsigned
7	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Lock and Unlock Range in Stream File (QHFLULSF) API locks or unlocks a range of bytes in an open stream file. An application can lock part of a file instead of the entire file to temporarily keep other open instances from accessing that part.

An open instance is the state of a file having been opened and assigned an open file handle. The same job can open a file more than once. At each open operation, the file is assigned a unique open file handle, and the system treats the resulting state of being open as unique. Locks obtained during one open instance are honored by other open instances, even when the later open instances occur during the same job.

A locked range can be located anywhere in a file, and locking beyond the end of the file is allowed. Locks on a range are independent of the lock mode specified when the file is opened with the Open Stream File (QHFOFNSF) API.

Once a lock is obtained, the specified access is denied to all other requests to access that range in the file. The specified access is denied until the range is explicitly unlocked, the file is explicitly closed by the job, or the file is implicitly closed when the job ends. Closing a file releases all locks on the file.

If both unlocking and locking are specified on the request, the range is unlocked first. When unlocking is complete, the range is locked. This allows one open instance to keep other open instances from using a range that must be unlocked and relocked.

Your application should keep track of the ranges it locks. The QHFLULSF API does not track them, and HFS does not provide an API that lists locks.

## Required Parameter Group

### Open file handle

INPUT; CHAR(16)

The file handle returned when the file was opened with the QHFOFNSF API.

### Lock information

INPUT; CHAR(6)

Additional information specifying the action to take. Valid values for the 6 characters in this parameter are:

1	The lock mode, indicating what access other open instances can have to this range of the file. Valid values are:
0	No lock. Use this when requesting an unlock operation only.
2	Deny write. Other open instances have read-only access to the locked range. The range can overlap or include other ranges locked in deny write mode, but it cannot overlap or include other ranges locked in deny read/write mode.
4	Deny read/write. This is an exclusive lock mode that denies other open instances all access to the locked range. The range cannot overlap or include any other locked range.
2-6	Reserved. These characters must be blank.

**File offset where lock begins**

INPUT; BINARY(4) UNSIGNED

The number of bytes from the beginning of the file where the range to lock begins.

**Bytes to lock**

INPUT; BINARY(4) UNSIGNED

The number of bytes to lock. If this is an unlock operation only, use zero for this parameter.

**File offset where unlock begins**

INPUT; BINARY(4) UNSIGNED

The number of bytes from the beginning of the file where the range to unlock begins.

**Bytes to unlock**

INPUT; BINARY(4) UNSIGNED

The number of bytes to unlock. If this is a lock operation only, use zero for this parameter.

**Error code**

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

## Error Messages

Message ID	Error Message Text
CPF1F2E E	Range of bytes in file in use.
CPF1F2F E	Unlock range of bytes in file failed.
CPF1F25 E	File handle not valid.
CPF1F28 E	Damaged file.
CPF1F32 E	Number of locks on file exceeds limit.
CPF1F4B E	Value for number of bytes not valid.
CPF1F4C E	Lock information value not valid.
CPF1F4D E	File offset value not valid.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F52 E	Error code not valid.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F82 E	Function not supported.
CPF1F87 E	Missing or damaged exit program &2.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

[Top](#) | ["Hierarchical File System APIs," on page 1](#) | [APIs by category](#)

---

## Open Stream File (QHFOFNSF) API

Required Parameter Group:	
1	Open file handle
<b>Output</b>	Char(16)
2	Path name
<b>Input</b>	Char(*)
3	Path name length
<b>Input</b>	Binary(4)
4	Open information
<b>Input</b>	Char(10)
5	Attribute information table
<b>Input</b>	Char(*)
6	Length of attribute information table
<b>Input</b>	Binary(4)
7	Action taken
<b>Output</b>	Char(1)
8	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Open Stream File (QHFOFNSF) API opens and optionally creates a single stream file. Applications can use the QHFOFNSF API to perform these tasks:

- Open an existing file.
- Open and replace an existing file. (You cannot perform this operation on read-only files)
- Create a new file and open it.
- Return an error if the specified file exists.
- Return an error if the specified file does not exist.

Do not use the QHFOFNSF API to change the directory entry attributes for an existing file. Instead, see “Change Directory Entry Attributes (QHFCGAT) API” on page 40.

When the file is opened, the file pointer is set to the first byte of the file (position 0). Subsequent read/write operations move or increase the value of the file pointer. To move it explicitly, see “Change File Pointer (QHFCGFP) API” on page 13.

### Required Parameter Group

#### Open file handle

OUTPUT; CHAR(16)

An identifier made up of arbitrary characters assigned by the API and used to refer to the file in subsequent operations.

**Path name**

INPUT; CHAR(\*)

The path name for the file. The last element of the path name is the file name.

**Path name length**

INPUT; BINARY(4)

The length of the path name, in bytes.

**Open information**

INPUT; CHAR(10)

Whether or not to open the file, and what the opened file's characteristics are. Each character of this parameter has a specific meaning. The characters and their meanings are:

1	The action to take if the file already exists. Valid values are:
	0 Do not open the file. Return an error.
	1 Open the file. When an existing file is opened and the file size is extended, the file system might not define the value of the new bytes.
	2 Replace the existing file. This is equivalent to deleting and re-creating the file. The directory entry information is replaced.
2	The action to take if the file does not exist. Valid values are:
	0 Return an error.
	1 Create the file.
3	The write-through flag for the file. Valid values are:
	0 Write operations to nonvolatile storage can be asynchronous. (Nonvolatile storage is any storage area whose contents are not lost when power is cut off or when the system is loaded.) An asynchronous write operation returns control to the application immediately, so it can continue the operation. The write operation occurs at a later, unspecified time.
	1 Write operations to nonvolatile storage must be synchronous. A <b>synchronous</b> write operation returns control to the operation only after the write operation completes.
4	Reserved. This field must be blank.

5 The file's lock mode. The **lock mode** defines what operations other jobs can perform on the file. Valid values are:

- 1 Deny None
- 2 Deny Write
- 3 Deny Read
- 4 Deny Read/Write (exclusive)

For a detailed description of lock modes, see "Lock and Access Modes" on page 25.

6 The file's **access mode**, indicating the application's access rights to the file. Valid values are:

- 0 Read Only
- 1 Write Only
- 2 Read/Write

For a detailed description of access modes, see "Lock and Access Modes" on page 25.

7 The type of open operation to perform. Valid values are:

- 0 Normal
- 1 Permanent. The file can be closed in only two ways: explicitly with the QHFCLOSF API, described in "Close Stream File (QHFCLOSF) API" on page 16, or implicitly when the job ends. The End Request (ENDRQS) and Reclaim Resource (RCLRSC) commands do not close the file.

8-10 Reserved. These characters must be blank.

### Attribute information table

INPUT; CHAR(\*)

The table specifying the attributes of the directory entry for this file. The file system determines which standard and extended attributes you can specify. For detailed descriptions of the standard attributes and the format of the table, see "HFS Attribute Information Table" on page 112.

Use this parameter only when creating a new file or replacing an existing file. It is ignored when opening an existing file.

### Length of the attribute information table

INPUT; BINARY(4)

The length of the attribute information table, in bytes, or a special value indicating which attributes to use. Valid values are:

- length* Use the attributes contained in the attribute information table.
- 0 Use the system defaults for standard attributes.



### Action taken

OUTPUT; CHAR(1)

One of these values, indicating the action taken by the file system:

- |   |  |
|---|--|
| 1 | The file already existed and was not replaced. |
| 2 | The file did not exist and was created.        |
| 3 | The file already existed and was replaced.     |

### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

## Lock and Access Modes

Lock and access modes determine which operations jobs can perform on files. The following sections describe lock and access modes in detail.

### Lock Modes

The lock mode determines the type of access your job lets other jobs have to the file. For example, if other jobs can continue reading a file but cannot write to it without impeding your job, specify deny write. This lock mode lets other jobs read the file but keeps them from writing to it.

When you assign a lock mode, it applies only to that specific occurrence of the file being opened. The lock mode you specified when opening a stream file restricts open operations by other jobs only; it does not restrict additional open operations by the job that locked the file.

A file can be opened multiple times by different jobs as long as the lock modes specified on the open operations are compatible.

Any locks placed on a file opened with the QHFOPNSF API are removed when the file is closed with the Close Stream File (QHFCLOSF) API or when the job ends.

The lock modes you can specify when opening a file are:

<i>Deny Read/Write</i>	Access to the file is exclusive for the current job. The current job can perform additional open operations on the file. However, no other job can open the file in any lock mode until the current job either closes it or ends.
<i>Deny Write</i>	Other jobs can read but cannot write to the file. In other words, no other job can open the file with write access; the file must be closed first, or the current job must end.
<i>Deny Read</i>	No other job can read the file until the current job either closes it or ends.
<i>Deny None</i>	Other jobs can read and write to the file. However, they cannot delete, rename, move, or change the attributes of the file until the current job either closes it or ends.

### Access Modes

The access mode characteristic determines the type of access your job needs to the file. For example, if your job requires read/write access and another job has already opened the file with a lock mode of deny none, your open request succeeds. However, if another job opened the file with a lock mode of deny write, your job is denied access.

The following table shows the results of opening and then trying to reopen the same file using all combinations of access and lock modes:

1st Open Operation (by your job)		2nd, 3rd, 4th Open Operation (by other jobs)											
Lock Mode	Access Mode	Deny Read/Write			Deny Write			Deny Read			Deny None		
		R/O	R/W	W/O	R/O	R/W	W/O	R/O	R/W	W/O	R/O	R/W	W/O
Deny Read/Write	R/O	N	N	N	N	N	N	N	N	N	N	N	N
	R/W	N	N	N	N	N	N	N	N	N	N	N	N
	W/O	N	N	N	N	N	N	N	N	N	N	N	N
Deny Write	R/O	N	N	N	Y	N	N	N	N	N	Y	N	N
	R/W	N	N	N	N	N	N	N	N	N	Y	N	N
	W/O	N	N	N	N	N	N	Y	N	N	Y	N	N
Deny Read	R/O	N	N	N	N	N	Y	N	N	N	N	N	Y
	R/W	N	N	N	N	N	N	N	N	N	N	N	Y
	W/O	N	N	N	N	N	N	N	N	Y	N	N	Y
Deny None	R/O	N	N	N	Y	Y	Y	N	N	N	Y	Y	Y
	R/W	N	N	N	N	N	N	N	N	N	Y	Y	Y
	W/O	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y

**Key:**

Y      Open is allowed  
N      Open is denied  
R/W    Read/write  
R/O    Read only  
W/O    Write only

## Error Messages

Message ID	Error Message Text
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F2A E	Number of open files exceeds limit.
CPF1F21 E	File name not valid.
CPF1F22 E	File not found.
CPF1F24 E	File name already exists.
CPF1F26 E	File in use.
CPF1F27 E	Authority not sufficient to access file.
CPF1F28 E	Damaged file.
CPF1F29 E	Use of reserved file name not allowed.
CPF1F37 E	File is a read-only file.

Message ID	Error Message Text
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F42 E	Attribute information table not valid.
CPF1F43 E	Attribute name not valid.
CPF1F44 E	Attribute value is not valid.
CPF1F46 E	Use of reserved attribute name not allowed.
CPF1F48 E	Path name not valid.
CPF1F49 E	Open information value not valid.
CPF1F52 E	Error code not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F81 E	API specific error occurred.
CPF1F82 E	Function not supported.
CPF1F83 E	File system name &1 not found.
CPF1F85 E	Not authorized to file system &1.
CPF1F87 E	Missing or damaged exit program &2.
CPF1F97 E	File system &1 in use.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

[Top](#) | ["Hierarchical File System APIs," on page 1](#) | [APIs by category](#)

---

## Read from Stream File (QHFRDSF) API

Required Parameter Group:	
1	Open file handle
<b>Input</b>	Char(16)
2	Data buffer
<b>Output</b>	Char(*)
3	Bytes to read
<b>Input</b>	Binary(4)
4	Bytes actually read
<b>Output</b>	Binary(4)
5	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Read from Stream File (QHFRDSF) API reads a specified number of bytes from a stream file opened with an access mode of read only or read/write.

The read operation starts at the current position of the **file pointer**, the location in the file where the next read or write operation occurs. When a file is opened with the Open Stream File (QHFOFNSF) API the file pointer is set to the first byte of the file (position 0). You can move the pointer explicitly with the Change File Pointer (QHFCHGFP) API; see page “Change File Pointer (QHFCHGFP) API” on page 13 for details. After the read operation, the file pointer value is increased by the number of bytes actually read.

## Required Parameter Group

### Open file handle

INPUT; CHAR(16)

The file handle returned when the file was opened with the QHFOFNSF API. Your application must have opened the file with an access mode of read only or read/write.

### Data buffer

OUTPUT; CHAR(\*)

The buffer that holds the data read from the file.

### Bytes to read

INPUT; BINARY(4)

The number of bytes to read from the file, starting at the current file pointer position. The number must be less than or equal to the length of the data buffer.

### Bytes actually read

OUTPUT; BINARY(4)

The number of bytes actually read and returned in the data buffer.

The value of this parameter equals the value of the bytes-to-read input parameter unless an error occurs or the end of the file is reached. Reaching the end of the file is not an error. When the file pointer reaches the end of the file, the file system stores the bytes actually read in the data buffer and sets the actual number of bytes read, which in this case is less than the number of bytes to read. The application can then detect the end of the file by comparing the number of bytes actually read to the number requested.

### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

## Error Messages

### Message ID    Error Message Text

CPF1F2C E	Read operation not allowed to file opened for write only.
CPF1F2E E	Range of bytes in file in use.
CPF1F25 E	File handle not valid.
CPF1F28 E	Damaged file.
CPF1F35 E	Read file operation failed.
CPF1F4B E	Value for number of bytes not valid.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F52 E	Error code not valid.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.

**Message ID    Error Message Text**

CPF1F73 E    Not authorized to use command.  
 CPF1F74 E    Not authorized to object.  
 CPF1F82 E    Function not supported.  
 CPF1F87 E    Missing or damaged exit program &2.  
 CPF3C90 E    Literal value cannot be changed.  
 CPF9872 E    Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

Top | "Hierarchical File System APIs," on page 1 | APIs by category

## Set Stream File Size (QHFSETSZ) API

Required Parameter Group:	
<b>1</b>	Open file handle
<b>Input</b>	Char(16)
<b>2</b>	File size
<b>Input</b>	Binary(4) Unsigned
<b>3</b>	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Set Stream File Size (QHFSETSZ) API sets the size of a stream file in bytes. Applications can use the QHFSETSZ API to increase or decrease the size of a stream file that has been opened with write only or read/write access.

Existing locks on the file are maintained. If the entire file is locked, you cannot change its size. If part of the file is locked, the new size cannot interfere with the locked part. The description of the file size parameter discusses these restrictions.

### Required Parameter Group

#### Open file handle

INPUT; CHAR(16)

The handle returned from the Open Stream File (QHFOPNSF) API when the file was opened. The file must have been opened with write only or read/write access.

#### File size

INPUT; BINARY(4) UNSIGNED

The size to make the file, in bytes. The size cannot start or end within a range of bytes locked in deny write or deny read/write mode, and it cannot extend beyond a locked range. If a locked range was not previously within the file, the file cannot be extended to include that range. If a locked range is within the file, the file cannot be truncated to exclude that range.

When the file size is increased, the file system may not define the value of the new bytes. When the file size is decreased, the data in the truncated part of the file is lost.

## Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

## Error Messages

### Message ID Error Message Text

CPF1F2B E	Write operation not allowed to file opened for read only.
CPF1F2E E	Range of bytes in file in use.
CPF1F25 E	File handle not valid.
CPF1F28 E	Damaged file.
CPF1F4B E	Value for number of bytes not valid.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F52 E	Error code not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F82 E	Function not supported.
CPF1F87 E	Missing or damaged exit program &2.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

[Top](#) | ["Hierarchical File System APIs," on page 1](#) | [APIs by category](#)

---

## Write to Stream File (QHFWRTSF) API

Required Parameter Group:

**1** Open file handle

**Input** Char(16)

**2** Data buffer

**Input** Char(\*)

**3** Bytes to write

**Input** Binary(4)

**4** Bytes actually written

**Output** Binary(4)

**5** Error code

**I/O** Char(\*)

Default Public Authority: \*USE

Threadsafe: No

The Write to Stream File (QHFWRTSF) API writes bytes to a stream file. The file must have been opened with an access mode of write only or read/write. If there is a deny write or deny read/write lock on a byte range being written to, the function fails.

The write operation starts at the current position of the file pointer, the location in the file where the next read or write operation occurs. When a file is opened with the Open Stream File (QHFOPNSF) API the file pointer is set to the first byte of the file. You can move the pointer explicitly with the QHFCHGFP API; see “Change File Pointer (QHFCHGFP) API” on page 13 for details. After the write operation, the file pointer value is increased by the number of bytes written.

## Required Parameter Group

### Open file handle

INPUT; CHAR(16)

The file handle returned when the file is opened with the QHFOPNSF API.

### Data buffer

INPUT; CHAR(\*)

The buffer containing the data being written.

### Bytes to write

INPUT; BINARY(4)

The number of bytes being written to the file, starting at the current file pointer position. The number must be less than or equal to the length of the data buffer.

The application can write beyond the end of the file. This increases the file’s size.

### Bytes actually written

OUTPUT; BINARY(4)

The number of bytes actually written to the file. If an error occurs during the write operation, the value of this parameter can be less than the value of the bytes-to-write input parameter.

### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

## Error Messages

Message ID	Error Message Text
CPF1F2B E	Write operation not allowed to file opened for read only.
CPF1F2E E	Range of bytes in file in use.
CPF1F25 E	File handle not valid.
CPF1F28 E	Damaged file.
CPF1F34 E	Attempted write operation beyond file size limit.
CPF1F36 E	Write file operation failed.
CPF1F4B E	Value for number of bytes not valid.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F52 E	Error code not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.

**Message ID    Error Message Text**

- CPF1F72 E    Internal file system error occurred.
- CPF1F73 E    Not authorized to use command.
- CPF1F74 E    Not authorized to object.
- CPF1F82 E    Function not supported.
- CPF1F87 E    Missing or damaged exit program &2.
- CPF3C90 E    Literal value cannot be changed.
- CPF9872 E    Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

Top | "Hierarchical File System APIs," on page 1 | APIs by category

## Copy Stream File (QHFCPYSF) API

Required Parameter Group:	
1	Source file path name
<b>Input</b>	Char(*)
2	Source file path name length
<b>Input</b>	Binary(4)
3	Copy information
<b>Input</b>	Char(6)
4	Target file path name
<b>Input</b>	Char(*)
5	Target file path name length
<b>Input</b>	Binary(4)
6	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Copy Stream File (QHFCPYSF) API copies an existing stream file into another stream file and optionally renames the copy. The existing file being copied is called the source file. The copy, or the file that the source is copied into, is called the target file.

All file attributes except the revision date and time are copied from the source file to the target file. The file revision date and time are set to the current date and time. The file creation date and time stay as they are—that is, the source file’s creation date and time.

For restrictions about using the QHFCPYSF API with distributed data management (DDM), see the information on hierarchical file system support in the Distributed Data Management book.

### Required Parameter Group

**Source file path name**  
INPUT; CHAR(\*)



The path name of the source file (the file being copied). The last element of the path name is the source file name.

The source file must be accessible. No other job can have the source file open with a deny read or deny read/write lock.

**Source file path name length**

INPUT; BINARY(4)

The length of the source file path name, in bytes.

**Copy information**

INPUT; CHAR(6)

The type of copy operation being performed. The 6 characters of this parameter are:

- 1                    The action to take if the target file already exists. Valid values are:
  - 0            Do not replace the existing file.
  - 1            Replace the existing file with the copy.
  - 2            Add the copy to the end of the existing file.
- 2-6                Reserved. These characters must be blank.

**Target file path name**

INPUT; CHAR(\*)

The path name of the target file (the copy or the file that the source is copied into). The last element of the path name is the target file name.

If the target file has a different name from the source file, it can be in the same path as the source.

The target file must be accessible in write mode. It cannot be a read-only file, and another job cannot have it open with a deny write or deny read/write lock.

**Target file path name length**

INPUT; BINARY(4)

The length of the target file path name, in bytes.

**Error code**

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

## Error Messages

Message ID	Error Message Text
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F2A E	Number of open files exceeds limit.
CPF1F2E E	Range of bytes in file in use.
CPF1F21 E	File name not valid.
CPF1F22 E	File not found.
CPF1F23 E	New file name same as old file name.
CPF1F24 E	File name already exists.
CPF1F26 E	File in use.

<b>Message ID</b>	<b>Error Message Text</b>
CPF1F27 E	Authority not sufficient to access file.
CPF1F28 E	Damaged file.
CPF1F29 E	Use of reserved file name not allowed.
CPF1F34 E	Attempted write operation beyond file size limit.
CPF1F35 E	Read file operation failed.
CPF1F36 E	Write file operation failed.
CPF1F37 E	File is a read-only file.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F42 E	Attribute information table not valid.
CPF1F43 E	Attribute name not valid.
CPF1F44 E	Attribute value is not valid.
CPF1F46 E	Use of reserved attribute name not allowed.
CPF1F48 E	Path name not valid.
CPF1F51 E	Copy information value not valid.
CPF1F52 E	Error code not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F81 E	API specific error occurred.
CPF1F82 E	Function not supported.
CPF1F83 E	File system name &1 not found.
CPF1F84 E	Operation across file systems not allowed.
CPF1F85 E	Not authorized to file system &1.
CPF1F87 E	Missing or damaged exit program &2.
CPF1F97 E	File system &1 in use.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

[Top](#) | ["Hierarchical File System APIs," on page 1](#) | [APIs by category](#)

---

## Delete Stream File (QHFDLTSF) API

Required Parameter Group:	
1	Path name
<b>Input</b>	Char(*)
2	Path name length
<b>Input</b>	Binary(4)
3	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Delete Stream File (QHFDLTSF) API deletes a single stream file. Both the directory entry associated with the file and all data contained in the file object are deleted.

### Required Parameter Group

#### Path name

INPUT; CHAR(\*)

The path name for the file being deleted. The last element of the path name is the file name.

The file cannot be open or in use, and it cannot be a read-only file.

#### Path name length

INPUT; BINARY(4)

The length of the path name, in bytes.

#### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

### Error Messages

Message ID	Error Message Text
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F21 E	File name not valid.
CPF1F22 E	File not found.
CPF1F26 E	File in use.
CPF1F27 E	Authority not sufficient to access file.
CPF1F28 E	Damaged file.
CPF1F37 E	File is a read-only file.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F48 E	Path name not valid.
CPF1F52 E	Error code not valid.

**Message ID    Error Message Text**

CPF1F61 E    No free space available on media.  
 CPF1F62 E    Requested function failed.  
 CPF1F63 E    Media is write protected.  
 CPF1F66 E    Storage needed exceeds maximum limit for user profile &1.  
 CPF1F71 E    Exception specific to file system occurred.  
 CPF1F72 E    Internal file system error occurred.  
 CPF1F73 E    Not authorized to use command.  
 CPF1F74 E    Not authorized to object.  
 CPF1F75 E    Error occurred during start-job-session function.  
 CPF1F81 E    API specific error occurred  
 CPF1F82 E    Function not supported.  
 CPF1F83 E    File system name &1 not found.  
 CPF1F85 E    Not authorized to file system &1.  
 CPF1F87 E    Missing or damaged exit program &2.  
 CPF1F97 E    File system &1 in use.  
 CPF3C90 E    Literal value cannot be changed.  
 CPF9872 E    Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## Move Stream File (QHFMVVSF) API

Required Parameter Group:

<b>1</b>	Source file path name
<b>Input</b>	Char(*)
<b>2</b>	Source file path name length
<b>Input</b>	Binary(4)
<b>3</b>	Target file path name
<b>Input</b>	Char(*)
<b>4</b>	Target file path name length
<b>Input</b>	Binary(4)
<b>5</b>	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE

Threadsafe: No

The Move Stream File (QHFMVVSF) API moves a single stream file from one directory to another and optionally changes the file’s name. The file’s attributes are not changed.

The QHFMVVSF API only moves stream files to a different path. To rename files within a path, see “Rename Stream File (QHFRNMSF) API” on page 38.

## Required Parameter Group

### Source file path name

INPUT; CHAR(\*)

The path name of the file being moved. The file name is the last element of the path name.

The source file must be accessible. You cannot move a file that is already in use by another job.

### Source file path name length

INPUT; BINARY(4)

The length of the source file path name, in bytes.

### Target file path name

INPUT; CHAR(\*)

The path name designating the new location and, optionally, the new name of the file being moved.

The target file cannot already exist. It must reside in a different path from the source file. The file name can be the same as or different from the source file name.

### Target file path name length

INPUT; BINARY(4)

The length of the target file path name, in bytes.

### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

## Error Messages

Message ID	Error Message Text
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F03 E	New directory name same as old directory name.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F2A E	Number of open files exceeds limit.
CPF1F2E E	Range of bytes in file in use.
CPF1F21 E	File name not valid.
CPF1F22 E	File not found.
CPF1F24 E	File name already exists.
CPF1F26 E	File in use.
CPF1F27 E	Authority not sufficient to access file.
CPF1F28 E	Damaged file.
CPF1F29 E	Use of reserved file name not allowed.
CPF1F34 E	Attempted write operation beyond file size limit.
CPF1F35 E	Read file operation failed.
CPF1F36 E	Write file operation failed.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F42 E	Attribute information table not valid.
CPF1F43 E	Attribute name not valid.
CPF1F44 E	Attribute value is not valid.
CPF1F46 E	Use of reserved attribute name not allowed.
CPF1F48 E	Path name not valid.

Message ID	Error Message Text
CPF1F52 E	Error code not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F81 E	API specific error occurred.
CPF1F82 E	Function not supported.
CPF1F83 E	File system name &1 not found.
CPF1F84 E	Operation across file systems not allowed.
CPF1F85 E	Not authorized to file system amp;1.
CPF1F87 E	Missing or damaged exit program &2.
CPF1F97 E	File system &1 in use.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## Rename Stream File (QHFRNMSF) API

Required Parameter Group:	
1	Path name
<b>Input</b>	Char(*)
2	Path name length
<b>Input</b>	Binary(4)
3	New file name
<b>Input</b>	Char(*)
4	New file name length
<b>Input</b>	Binary(4)
5	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Rename Stream File (QHFRNMSF) API renames a stream file in the same path.

### Required Parameter Group

**Path name**  
INPUT; CHAR(\*)

The path name for the file being renamed. The last element of the path name is the name of the file.

The file cannot be open or in use.

#### **Path name length**

INPUT; BINARY(4)

The length of the path name, in bytes.

#### **New file name**

INPUT; CHAR(\*)

The new name for the file. Do not include the path.

The new name must be unique within a directory. A file with this name cannot already exist in the target directory.

#### **New file name length**

INPUT; BINARY(4)

The length of the new file name, in bytes.

#### **Error code**

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

## **Error Messages**

<b>Message ID</b>	<b>Error Message Text</b>
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F21 E	File name not valid.
CPF1F22 E	File not found.
CPF1F23 E	New file name same as old file name.
CPF1F24 E	File name already exists.
CPF1F26 E	File in use.
CPF1F27 E	Authority not sufficient to access file.
CPF1F28 E	Damaged file.
CPF1F29 E	Use of reserved file name not allowed.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F48 E	Path name not valid.
CPF1F52 E	Error code not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F81 E	API specific error occurred.
CPF1F82 E	Function not supported.
CPF1F83 E	File system name &1 not found.
CPF1F85 E	Not authorized to file system &1.

**Message ID    Error Message Text**

CPF1F87 E    Missing or damaged exit program &2.  
 CPF1F97 E    File system &1 in use.  
 CPF3C90 E    Literal value cannot be changed.  
 CPF9872 E    Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

Top | "Hierarchical File System APIs," on page 1 | APIs by category

## Change Directory Entry Attributes (QHFCHGAT) API

Required Parameter Group:	
1	Path name
<b>Input</b>	Char(*)
2	Path name length
<b>Input</b>	Binary(4)
3	Attribute information table
<b>Input</b>	Char(*)
4	Length of the attribute information table
<b>Input</b>	Binary(4)
5	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Change Directory Entry Attributes (QHFCHGAT) API changes the attributes of a specified directory entry for an existing file or directory. As long as no other job has opened the directory in which the file or directory is located with a deny write lock, your application can add, change, or delete directory entry attributes.

### Required Parameter Group

#### Path name

INPUT; CHAR(\*)

The path name of the directory or file whose attributes you want to change. The directory or file must exist, and the path name must have more than one element. You cannot change the directory entry attributes of a file system.

#### Path name length

INPUT; BINARY(4)

The length of the path name, in bytes.

#### Attribute information table

INPUT; CHAR(\*)



The table specifying the directory entry attributes to change. The file system determines which standard and extended attributes you can specify. For descriptions of the standard attributes, see “HFS Directory Entry Attributes” on page 109. For the format of the table, see “HFS Attribute Information Table” on page 112.

#### **Length of the attribute information table**

INPUT; BINARY(4)

The length of the attribute information table.

#### **Error code**

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

## **Error Messages**

<b>Message ID</b>	<b>Error Message Text</b>
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F21 E	File name not valid.
CPF1F22 E	File not found.
CPF1F26 E	File in use.
CPF1F27 E	Authority not sufficient to access file.
CPF1F28 E	Damaged file.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F42 E	Attribute information table not valid.
CPF1F43 E	Attribute name not valid.
CPF1F44 E	Attribute value is not valid.
CPF1F46 E	Use of reserved attribute name not allowed.
CPF1F48 E	Path name not valid.
CPF1F52 E	Error code not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F81 E	API specific error occurred.
CPF1F82 E	Function not supported.
CPF1F83 E	File system name &1 not found.
CPF1F85 E	Not authorized to file system &1.
CPF1F87 E	Missing or damaged exit program &2.
CPF1F97 E	File system &1 in use.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## Close Directory (QHFCLODR) API

Required Parameter Group:

1 Open directory handle

**Input** Char(16)

2 Error code

**I/O** Char(\*)

Default Public Authority: \*USE

Threadsafe: No

The Close Directory (QHFCLODR) API closes a specified directory that was opened using the Open Directory (QHFOPNDR) API. Once a directory is closed, the open directory handle that refers to it is no longer valid. If a process ends without closing directories, they are closed automatically. However, it is best if the application closes directories that it no longer needs so that other applications have free access to them.

### Required Parameter Group

#### Open directory handle

INPUT; CHAR(16)

The directory handle returned by the QHFOPNDR API when the directory was opened.

#### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

### Error Messages

Message ID	Error Message Text
------------	--------------------

CPF1F05 E	Directory handle not valid.
-----------	-----------------------------

CPF1F06 E	Directory in use.
-----------	-------------------

CPF1F08 E	Damaged directory.
-----------	--------------------

CPF1F41 E	Severe error occurred while addressing parameter list.
-----------	--

CPF1F52 E	Error code not valid.
-----------	-----------------------

CPF1F62 E	Requested function failed.
-----------	----------------------------

CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
-----------	---

CPF1F71 E	Exception specific to file system occurred.
-----------	---

CPF1F72 E	Internal file system error occurred.
-----------	--------------------------------------

CPF1F73 E	Not authorized to use command.
-----------	--------------------------------

CPF1F74 E	Not authorized to object.
-----------	---------------------------

CPF1F82 E	Function not supported.
-----------	-------------------------

CPF1F87 E	Missing or damaged exit program &2.
-----------	-------------------------------------

CPF3C90 E	Literal value cannot be changed.
-----------	----------------------------------

CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
-----------	--

API introduced: V2R1

Top | "Hierarchical File System APIs," on page 1 | APIs by category

---

## Open Directory (QHFOPNDR) API

Required Parameter Group:	
1	Directory handle
<b>Output</b>	Char(16)
2	Path name
<b>Input</b>	Char(*)
3	Path name length
<b>Input</b>	Binary(4)
4	Open information
<b>Input</b>	Char(6)
5	Attribute selection table
<b>Input</b>	Char(*)
6	Length of attribute selection table
<b>Input</b>	Binary(4)
7	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Open Directory (QHFOPNDR) API opens the specified directory so its directory entries can be read. At open time, the directory pointer points to the first entry in the directory. As directory entries are read using the Read Directory Entries (QHFRDDR) API, the directory pointer advances so that the next entries will be read during future read operations.

Opening a directory can help streamline information retrieval and protect the directory. For example, you might open a directory to:

- Improve performance when obtaining information from more than one directory entry.
- Prevent the directory from being renamed or deleted. A deny none lock mode lets other processes read or change the contents of the directory, but keeps them from renaming or deleting the directory itself.
- Prevent the contents of the directory from being changed. A deny write lock mode keeps other processes from adding to or otherwise changing the open directory's contents. Directories and files cannot be created in the directory until it is closed.

### Required Parameter Group

#### Directory handle

OUTPUT; CHAR(16)

An identifier made up of arbitrary characters returned by the API and used to identify the directory for subsequent operations, such as reading and closing.

#### Path name

INPUT; CHAR(\*)

The path name for the directory being opened. The path and directory must exist.

For the directory name (the last element of the path name), you can use either a specific name or a generic name.

If the last element in the path is a specific name, that directory is opened and all directory entries in the directory are available for subsequent read operations.

If the last element in the path is a generic name, it identifies the directory entries to make available for subsequent read operations; the previous name in the path specifies the directory to open. Directory entries that are in the directory to open and that match the generic name are made available.

You can use these special matching characters in generic names:

- \* An asterisk stands for zero or more characters. You can use it anywhere in a string.
- ? A question mark at the end of a string represents zero or one character. A question mark embedded in a string represents one character.

For example, /QDLS/BUSY/DEPT\* indicates all directories and files that have names beginning with DEPT and that are located in directory BUSY in the QDLS file system.

#### Path name length

INPUT; BINARY(4)

The length of the path name, in bytes.

#### Open information

INPUT; CHAR(6)

Information about the type and mode of the open operation. The characters and their meanings are:

- 1 The lock mode, indicating what other jobs can do to the directory. Valid values are:
  - 0 No lock. Other jobs can read, change, rename, or delete the directory.
  - 1 Deny none. Other jobs can read or change directory entries, but they cannot rename or delete the directory.
  - 2 Deny write. Other jobs can read the contents of the directory, but they cannot change, rename, or delete it.
- 2 The type of open operation to perform. Valid values are:
  - 0 Normal.
  - 1 Permanent. The directory can be closed in only two ways, explicitly by the Close Directory (QHFCLODR) API, or implicitly, when the job ends. End request and reclaim resource operations do not close the directory.
- 3-6 Reserved. These characters must be blank.

#### Attribute selection table

INPUT; CHAR(\*)

The table specifying which attributes are available when reading directory entries. The file system determines which standard and extended attributes you can specify. For detailed descriptions of the standard attributes, see “HFS Directory Entry Attributes” on page 109. For the format of the table, see “HFS Attribute Selection Table” on page 113.

This parameter lets you choose which *attributes* of the directory entries are available for reading when the directory is open. It does not determine which *directory entries* can be read. Use the path name parameter to select the directory entries you want to read.

#### Length of the attribute selection table

INPUT; BINARY(4)

The length of the table, in bytes, or a special value indicating which attributes are made available. Valid values are:

<i>length</i>	The attribute selection table parameter contains the attributes the application wants to make available.
0	Only the standard attribute QNAME is available.
-1	All attributes are available.

#### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

## Error Messages

Message ID	Error Message Text
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F43 E	Attribute name not valid.
CPF1F45 E	Attribute selection table not valid.
CPF1F48 E	Path name not valid.
CPF1F49 E	Open information value not valid.
CPF1F52 E	Error code not valid.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F81 E	API specific error occurred.
CPF1F82 E	Function not supported.
CPF1F83 E	File system name &1 not found.
CPF1F85 E	Not authorized to file system &1.
CPF1F87 E	Missing or damaged exit program &2.
CPF1F97 E	File system &1 in use.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

## Read Directory Entries (QHFRDDR) API

Required Parameter Group:	
1	Open directory handle
<b>Input</b>	Char(16)
2	Data buffer
<b>Output</b>	Char(*)
3	Data buffer length
<b>Input</b>	Binary(4)
4	Number of directory entries to read
<b>Input</b>	Binary(4)
5	Number of directory entries read
<b>Output</b>	Binary(4)
6	Length of data returned
<b>Output</b>	Binary(4)
7	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Read Directory Entries (QHFRDDR) API reads one or more directory entries from a directory opened with the Open Directory (QHFOPNDR) API. The QHFOPNDR API's path name parameter determines which directory entries are read. The QHFOPNDR API's attribute selection table determines what information is returned for each directory entry. For details about the QHFOPNDR API, see "Open Directory (QHFOPNDR) API" on page 43.

You must open a directory before reading from it. The open directory handle returned by the QHFOPNDR API is needed as input to the QHFRDDR API.

The QHFRDDR API reads directory entries sequentially. Each time the QHFRDDR API is called, the directory pointer is advanced by the number of directory entries returned in the data buffer. Subsequent calls of the QHFRDDR API return additional directory entries, until there are no more to return.

### Required Parameter Group

#### Open directory handle

INPUT; CHAR(16)

The directory handle obtained when the directory was opened with the QHFOPNDR API.

#### Data buffer

OUTPUT; CHAR(\*)

The buffer to hold the directory entry information returned. For the format, see “Data Buffer.”

**Data buffer length**

INPUT; BINARY(4)

The length of the data buffer described in “Data Buffer.” The buffer must be large enough to hold the requested attributes for at least one directory entry. If it is too small, the read operation fails and no data is returned. However, the length of data returned parameter contains the total number of bytes the file system tried to return for the next directory entry. The application should increase the data buffer size to at least that number and try the request again.

**Number of directory entries to read**

INPUT; BINARY(4)

The number of directory entries to place in the data buffer.

**Number of directory entries read**

OUTPUT; BINARY(4)

The number of directory entries actually placed in the data buffer. The value of this field is 0 when there are no more directory entries to read.

**Length of data returned**

OUTPUT; BINARY(4)

If the read operation is successful, this field contains the total number of bytes returned in the data buffer. If the read operation is not successful because the data buffer is not large enough to hold at least one entry, this field contains the number of bytes required to hold the requested attributes for the next directory entry.

**Error code**

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

**Data Buffer**

The data buffer holds the information returned about each directory entry. What information is returned depends mostly on what attributes are selected in the attribute selection table when the directory is opened with the QHFOPNDR API. However, the QNAME attribute is returned with every directory entry even though it is never specified in the attribute selection table. Thus, the data buffer always contains at least one piece of information about each directory entry found, its name. If the directory entry exists but cannot be read because of damage or a lock by another process, two pieces of information are returned: the name, given in the QNAME attribute, and the error that occurred, given in the QERROR attribute. For details about these attributes, see “HFS Directory Entry Attributes” on page 109.

The data buffer has three logical parts:

1. The first field specifies the number of directory entries returned.
2. The next fields give the offsets to the directory entries returned. There is one offset field for each directory entry.
3. Next are the attribute information tables for the directory entries returned. There is one attribute information table for each directory entry.

The following table shows the format of the data buffer. The offset fields are repeated until the offsets for all directory entries are listed; the attribute information table for each directory entry is repeated in the same way.

The format of the data buffer is:

Type	Field
BINARY(4)	Number of directory entries returned.
BINARY(4)	Offset to the first directory entry.
BINARY(4)	Offset to the next directory entry, if more than one exists. This field is repeated for each directory entry returned.
Directory entry	Attribute information table for the first directory entry.
Directory entry	Attribute information table for the next directory entry, if more than one exists. This field is repeated for each directory entry returned.

**Note:** Each directory entry in the table is represented by the standard attribute information table described in HFS Attribute Information Table. Offsets within the directory entries are from the beginning of the directory entry, not from the beginning of the data buffer.

## Error Messages

Message ID	Error Message Text
CPF1F05 E	Directory handle not valid.
CPF1F08 E	Damaged directory.
CPF1F4A E	Value for number of directory entries not valid.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F47 E	Buffer overflow occurred.
CPF1F52 E	Error code not valid.
CPF1F53 E	Value for length of data buffer not valid.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F82 E	Function not supported.
CPF1F87 E	Missing or damaged exit program &2.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)



---

## Retrieve Directory Entry Attributes (QHFRTVAT) API

Required Parameter Group:	
1	Path name
<b>Input</b>	Char(*)
2	Path name length
<b>Input</b>	Binary(4)
3	Attribute selection table
<b>Input</b>	Char(*)
4	Length of attribute selection table
<b>Input</b>	Binary(4)
5	Attribute information table
<b>Output</b>	Char(*)
6	Length of attribute information table
<b>Input</b>	Binary(4)
7	Length of data returned
<b>Output</b>	Binary(4)
8	Error code
<b>I/O</b>	Char(*)
	Default Public Authority: *USE
Threadsafe: No	

The Retrieve Directory Entry Attributes (QHFRTVAT) API retrieves attribute information from a specified directory entry for a directory or file. The QHFRTVAT API might be faster and more efficient than explicitly opening, reading, and then closing the directory, even if your file system automatically opens and closes the directory during its retrieve operation.

You can use the QHFRTVAT API to determine whether a specific directory entry exists, as well as to get one or more attributes of a specific directory entry. The QHFRTVAT API works with only one directory entry at a time. To retrieve the attributes of several directory entries at once, see “Read Directory Entries (QHFRTDDR) API” on page 46 and “Open Directory (QHFOPNDR) API” on page 43.

### Required Parameter Group

#### Path name

INPUT; CHAR(\*)

The path name of the directory or file to retrieve attributes from. The directory or file must exist, and the path name must have more than one element. You cannot retrieve directory entry attributes for a file system.

#### Path name length

INPUT; BINARY(4)

The length of the path name, in bytes.

### Attribute selection table

INPUT; CHAR(\*)

The table specifying the attributes to be returned in the attribute information table. The file system determines which standard and extended attributes you can specify. For descriptions of the standard attributes, see "HFS Directory Entry Attributes" on page 109. For the format of the table, see "HFS Attribute Selection Table" on page 113.

### Length of the attribute selection table

INPUT; BINARY(4)

The length of the attribute selection table, in bytes, or a special value indicating which attributes are returned. Valid values are:

*length*

The attribute selection table parameter contains the attributes the application wants to make available.

0

No attributes are returned. You can use this to see whether the directory entry exists.

-1

All attributes are returned.

### Attribute information table

OUTPUT; CHAR(\*)

The directory entry information returned, as specified in the attribute selection table parameter. For the format of the table containing the returned information, see "HFS Attribute Information Table" on page 112.

### Length of the attribute information table

INPUT; BINARY(4)

The length of the attribute information table. The table must be large enough to hold all the attributes requested. If it is too small, the retrieve operation fails and no attribute information is returned; however, the length of data returned parameter contains the number of bytes the file system tried to return for that directory entry. The application should increase the attribute information table's length to at least that size and try the request again.

### Length of data returned

OUTPUT; BINARY(4)

If the retrieve operation is successful, this field contains the total number of bytes returned in the attribute information table.

If the retrieve operation fails because the attribute information table is too small to hold all of the attributes requested, this field contains the number of bytes required to hold the requested attributes.

### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

## Error Messages

Message ID	Error Message Text
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F21 E	File name not valid.

Message ID	Error Message Text
CPF1F22 E	File not found.
CPF1F26 E	File in use.
CPF1F27 E	Authority not sufficient to access file.
CPF1F28 E	Damaged file.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F42 E	Attribute information table not valid.
CPF1F43 E	Attribute name not valid.
CPF1F45 E	Attribute selection table not valid.
CPF1F47 E	Buffer overflow occurred.
CPF1F48 E	Path name not valid.
CPF1F52 E	Error code not valid.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F81 E	API specific error occurred.
CPF1F82 E	Function not supported.
CPF1F83 E	File system name &1 not found.
CPF1F85 E	Not authorized to file system &1.
CPF1F87 E	Missing or damaged exit program &2.
CPF1F97 E	File system &1 in use.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## File System Registration APIs

The file system registration APIs allow you to create support for the HFS APIs so that application programmers can use them with new hierarchical file systems you are creating or installing.

See the following to understand how to use the HFS APIs and exit programs with a new hierarchical file system.

- “New File Systems” on page 114
- “Enabling Your File System to HFS” on page 115
- “How HFS Support Processes a File System Job” on page 116
- “Standard HFS API and Exit Program Functions” on page 117

The file system registration APIs are for registering and deregistering file system to HFS. The APIs are:

- “Deregister File System (QHFDGRGFS) API” on page 52 (QHFDGRGFS) reverses file system registration, preventing applications from using the file system through the HFS APIs.
- “Register File System (QHFRGFS) API” on page 54 (QHFRGFS) registers a file system with the HFS APIs so that application programmers can use the APIs to work with the file system.

The HFS exit programs support the HFS APIs. You must create exit programs to support these APIs.

The operation parameters used in these exit programs are followed by an abbreviation, the function name. For example, in the Start Job Session exit program, the initialize (INIT) function is being performed. The function names are passed to the HFS API so that the file system knows which operation is being performed.

The HFS exit programs are:

- “Exit Program for Change Directory Entry Attributes (QHFCHGAT) API” on page 58
- “Exit Program for Change File Pointer (QHFCHGFP) API” on page 60
- “Exit Program for Close Directory (QHFCLODR) API” on page 62
- “Exit Program for Close Stream File (QHFCLOSF) API” on page 64
- “Exit Program for Control File System (QHFCTLFS) API” on page 66
- “Exit Program for Copy Stream File (QHFCPYSF) API” on page 68
- “Exit Program for Create Directory (QHFCRTDR) API” on page 73
- “Exit Program for Delete Directory (QHFDLTD) API” on page 75
- “Exit Program for Delete Stream File (QHFDLTFS) API” on page 77
- “End Job Session Exit Program” on page 79
- “Exit Program for Force Buffered Data (QHFFRCSF) API” on page 79
- “Exit Program for Get Stream File Size (QHFGGETSZ) API” on page 81
- “Exit Program for Lock and Unlock Range in Stream File (QHFLULSF) API” on page 83
- “Exit Program for Move Stream File (QHFMVMSF) API” on page 85
- “Exit Program for Open Directory (QHFOPNDR) API” on page 88
- “Exit Program for Open Stream File (QHFOPNFS) API” on page 90
- “Exit Program for Read Directory Entries (QHFRDDR) API” on page 93
- “Exit Program for Read from Stream File (QHFRDSF) API” on page 95
- “Exit Program for Rename Directory (QHFRNMDR) API” on page 97
- “Exit Program for Rename Stream File (QHFRNMSF) API” on page 99
- “Exit Program for Retrieve Directory Entry Attributes (QHFRTVAT) API” on page 101
- “Exit Program for Set Stream File Size (QHFSSETSZ) API” on page 103
- “Start Job Session Exit Program” on page 105
- “Exit Program for Write to Stream File (QHFWRFSF) API” on page 106

“Hierarchical File System APIs,” on page 1 | APIs by category

---

## Deregister File System (QHFDRGFS) API

Required Parameter Group:

1	File system name
<b>Input</b>	Char(10)
2	Error code
<b>I/O</b>	Char(*)
	Threadsafe: No

The Deregister File System (QHFDRGFS) API removes a file system and its functions from HFS support so that applications can no longer work with the file system through the HFS APIs. You can use the QHFDRGFS API to keep users from working with a file system while you upgrade to a new release.

If there are open files or directories in the file system being deregistered, HFS support automatically closes them before deregistering the file system. See “End Job Session Exit Program” on page 79 for details.

## Required Parameter Group

### File system name

INPUT; CHAR(10);

The name of the file system being deregistered.

**Note:** You cannot deregister the document library services (DLS) file system, QDLS.

For deregistration to succeed, the file system cannot be in use. If a job that called the file system is not yet complete, the file system’s Start Job Session exit program is still locked on behalf of that job, and the file system is still in use.

### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

## Error Messages

### Message ID    Error Message Text

CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F52 E	Error code not valid.
CPF1F81 E	API specific error occurred.
CPF1F85 E	Not authorized to file system &1.
CPF1F87 E	Missing or damaged exit program &2.
CPF1F9B E	Reregister or deregister file system failed.
CPF1F92 E	File system &1 not registered.
CPF1F97 E	File system &1 in use.
CPF1F98 E	Registration or deregistration cannot be done now.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## Register File System (QHFRGFS) API

Required Parameter Group:	
1	File system name
<b>Input</b>	Char(10)
2	Version
<b>Input</b>	Char(6)
3	Registration information
<b>Input</b>	Char(6)
4	List of qualified exit program names
<b>Input</b>	Array of Char(20)
5	File system description
<b>Input</b>	Char(50)
6	Error code
<b>I/O</b>	Char(*)
	Threadsafe: No

The Register File System (QHFRGFS) API adds a new file system to OS/400 HFS support and records the APIs that it supports so they are accessible to user applications. You must register a file system before users can access it with the HFS APIs.

### Authorities and Locks

#### *Exit Program Authority*

\*USE or higher for all exit programs being registered.

#### *File System Authority*

For information about users' authority to use the file systems you register, see User Authorizations and Locks for File System Functions.

### Required Parameter Group

#### **File system name**

INPUT; CHAR(10)

The name of the file system being registered. The name can be 1-10 characters long. The first character must be a capital letter other than Q (Q is reserved for IBM-supplied file systems). The remaining characters can be any combination of capital letters (A-Z) and numbers (0-9). The name cannot contain lowercase letters (a-z), special characters, or quotation marks.

The file system name determines where calls to HFS APIs are sent. For example, a call that contains a path name beginning with NEWS, such as /NEWS/DIRA/FILE1, is sent to the NEWS file system.

**Note:** IBM preregisters the document library services (DLS) file system, QDLS. You cannot register or deregister it yourself.

#### **Version**

INPUT; CHAR(6)

The version indicates the level of HFS the file system chooses to use. Use the format VxRxMx where x stands for the version, release, and modification levels, respectively. The values indicate the different versions in which HFS enabled new support that the file systems can use. The valid versions are:

V2R1M0          Version 2 Release 1 Modification Level 0  
V2R3M0          Version 2 Release 3 Modification Level 0

## Registration information

INPUT; CHAR(6)

Additional information describing the actions to take during registration.

The characters in this parameter are:

- |   |  |
|---|--|
| 1 | Whether to register a file system that is already registered. This character lets you reregister a changed file system without deregistering it first. Valid values are:   |
|   | <p>0          Do not reregister the file system.</p> <p>1          Reregister the file system.</p>   |
| 2 | Which type of cross-file-system copy or move operation to perform. This character is called the <b>copy or move indicator</b> . HFS support checks its value only when an application specifies different source and target file systems in calls to the Copy Stream File (QHFCPYSF) or Move Stream File (QHFMVVSF) API.   |
|   | <p>This character has no effect on operations within the same file system. For cross-file-system operations, it tells the QHFCPYSF and QHFMVVSF APIs whether to call the source file system's copy and move exit programs to see if they can perform the operations. If they cannot, the API tries the exit programs for the target file system. If the source file system and the target file system cannot perform the operations, the API calls a series of other exit programs (such as those that open, read from, and write to stream files) to perform the operation. The last method is the least efficient.</p> <p>For a detailed explanation of cross-file-system copy and move processing, see "Exit Program for Copy Stream File (QHFCPYSF) API" on page 68.</p> <p>Valid values for this character are:</p> <p>0          Do not call this file system's Copy Stream File or Move Stream File exit program when performing cross-file-system operations. The copy and move exit programs for this file system cannot communicate directly with any other file system, so the APIs should not waste time trying them.</p> <p>1          Call this file system's Copy Stream File or Move Stream File exit program when performing cross-file-system operations. This file system's copy and move exit programs might be able to perform cross-file-system operations in some cases, so the APIs should try them before trying the less efficient copy and move method.</p> |

**List of qualified exit program names**

INPUT; ARRAY OF CHAR(20)

An array listing the exit programs that perform the work for the HFS APIs. (An **array** is a list of items in a specific sequence.) The first 10 characters of each array element contain the exit program name, and the second 10 characters of each array element contain the name of the library in which the exit program resides.

If the file system being registered does not support a particular API and thus there is no exit program, specify \*NONE for the program name and blanks for the library name. If an application calls an API for which there is no exit program, the API issues a message stating that the file system does not support that operation.

If the file system is written in a language that supports a variable number of input parameters, you can specify the same exit program for more than one function.

The sequence of array elements indicates the operation or API supported by the exit program specified there. For example, the exit program you specify in position 3 is called when an application calls the QHFCRTDR API. The sequence is as follows:

1. Start Job Session Operation (*required*)
2. End Job Session Operation (*required*)
3. Create Directory (for the QHFCRTDR API)
4. Open Directory (for the QHFOPNDR API)
5. Read Directory Entries (for the QHFRDDR API)
6. Close Directory (for the QHFCLODR API; *required if an Open Directory exit program is specified*)
7. Retrieve Directory Entry Attributes (for the QHFRTVAT API)
8. Change Directory Entry Attributes (for the QHFCHGAT API)
9. Delete Directory (for the QHFDLTDR API)
10. Rename Directory (for the QHFRNMDR API)
11. Open Stream File (for the QHFOPNSF API)
12. Read from Stream File (for the QHFRDSF API)
13. Write to Stream File (for the QHFWRTSF API)
14. Lock and Unlock Range in Stream File (for the QHFLULSF API)
15. Change File Pointer (for the QHFCHGFP API)
16. Force Buffered Data (for the QHFFRCSF API)
17. Get Stream File Size (for the QHFGETSZ API)
18. Set Stream File Size (for the QHFSETSZ API)
19. Close Stream File (for the QHFCLOSF API; *required if an Open Stream File exit program is specified*)
20. Copy Stream File (for the QHFCPYSF API)
21. Delete Stream File (for the QHFDLTSF API)
22. Move Stream File (for the QHFMOVSF API)
23. Rename Stream File (for the QHFRNMSF API)
24. Control File System (for the QHFCTLFS API)

**File system description**

INPUT; CHAR(50)



A brief description of the file system. This description is returned when applications use the List Registered File Systems (QHFLSTFS) or the Display Hierarchical File Systems (DSPHFS) command.

#### **Error code**

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

## **User Authorizations and Locks for File System Functions**

The exit program that performs the Start Job Session operation controls user authority and locking for the file system as a whole. The Start Job Session exit program is called the first time a job uses a given file system. At that time, the system checks the job's authority to the Start Job Session exit program. Authority for this program gives the user authority to use all other valid exit programs for the file system. OS/400 HFS support also obtains a shared read (\*SHRRD) lock on the exit program and maintains it until the end of the job so that other jobs cannot deregister the file system while it is in use. For details about this exit program, see "Start Job Session Exit Program" on page 105.

HFS support maintains system pointers to the file system's exit programs. HFS support uses these pointers when your exit programs are called. If a system pointer to any exit program becomes inaccurate, as it does when the program is recompiled, HFS support updates the pointer. If the system pointer cannot be resolved to the object, an error is returned.

You can change and recompile any exit program without having to reregister your file system. You should remember that if others are using the file system, they will have a \*SHRRD lock on the Start Job Session exit program. This may prevent recompilation of that program. You should only recompile the Start Job Session program when no one else is using the file system. You can use the Work with Object Locks (WRKOBJLCK) command on the Start Job Session program to see if any other job is using the file system.

## **Error Messages**

<b>Message ID</b>	<b>Error Message Text</b>
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F52 E	Error code not valid.
CPF1F74 E	Not authorized to object.
CPF1F81 E	API specific error occurred.
CPF1F85 E	Not authorized to file system &1.
CPF1F9A E	Exit program list not valid.
CPF1F9B E	Reregister or deregister file system failed.
CPF1F91 E	File system name not valid.
CPF1F93 E	File system &1 already registered.
CPF1F94 E	Exit program &2 not found.
CPF1F95 E	Required exit program not specified.
CPF1F96 E	Version level &2 not valid.
CPF1F97 E	File system &1 in use.
CPF1F98 E	Registration or deregistration cannot be done now.
CPF1F99 E	Register information value not valid.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R1

[Top](#) | ["Hierarchical File System APIs," on page 1](#) | [APIs by category](#)

---

## Exit Programs

These are the Exit Programs for this category.

---

### Exit Program for Change Directory Entry Attributes (QHFCHGAT) API

Required Parameter Group:	
1	Operation (CHGAT)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	Reserved
<b>Input</b>	Char(20)
4	Path name
<b>Input</b>	Char(*)
5	Path name length
<b>Input</b>	Binary(4)
6	Attribute information table
<b>Input</b>	Char(*)
7	Length of attribute information table
<b>Input</b>	Binary(4)

Before applications can use the Change Directory Entry Attributes (QHFCHGAT) API with your file system, you must:

1. Write an exit program that performs the change attribute operation on behalf of the API. For a detailed description of the API and its calling parameters, see “Change Directory Entry Attributes (QHFCHGAT) API” on page 40.
2. Give the exit program’s name when you register the file system with the Register File System API, QHFRGFS.

After that, when an application calls the QHFCHGAT API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

### Required Parameter Group

The API passes this information to your exit program:

#### Operation (CHGAT)

INPUT; CHAR(5)

The abbreviation for the operation being performed (CHGAT).

#### File system job handle

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

**Reserved**

INPUT; CHAR(20)

Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

**Path name**

INPUT; CHAR(\*)

The API removes the file system name before passing the path name to the exit program.

**Path name length**

INPUT; BINARY(4)

**Attribute information table**

INPUT; CHAR(\*)

**Length of the attribute information table**

INPUT; BINARY(4)

**API Functions**

The QHFCHGAT API performs the standard functions described in “Standard HFS API Functions” on page 117. The API does not validate the attribute information table in any way. It checks only the length parameter to make sure it has a valid value.

**Exit Program Requirements**

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118 and these additional functions:

- Validates the attribute information table.
- Changes valid attributes in the directory entry.
- Ignores a request to delete an attribute that does not exist for the directory entry.
- Ignores a request to change an attribute that cannot be changed or that does not apply to the directory entry. For example, the exit program must ignore requests to change the QFILSIZE attribute in a directory entry for a directory object and must ignore requests to change directories to files, or vice versa.

**Error Messages for Exit Program Use**

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F21 E	File name not valid.
CPF1F22 E	File not found.
CPF1F26 E	File in use.
CPF1F27 E	Authority not sufficient to access file.
CPF1F28 E	Damaged file.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F42 E	Attribute information table not valid.
CPF1F43 E	Attribute name not valid.
CPF1F44 E	Attribute value is not valid.
CPF1F46 E	Use of reserved attribute name not allowed.

Message ID	Error Message Text
CPF1F48 E	Path name not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

Top | "Hierarchical File System APIs," on page 1 | APIs by category

---

## Exit Program for Change File Pointer (QHFCHGFP) API

Required Parameter Group:	
1	Operation (CHGFP)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	Open file handle
<b>Input</b>	Char(16)
4	Move information
<b>Input</b>	Char(6)
5	Distance to move
<b>Input</b>	Binary(4)
6	New offset
<b>Output</b>	Binary(4) Unsigned

Before applications can use the Change File Pointer (QHFCHGFP) API with your file system, you must:

1. Write an exit program that performs the change file pointer operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Change File Pointer (QHFCHGFP) API" on page 13.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFCHGFP API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

## Required Parameter Group

The following shows the input parameters that the API passes to your exit program and the output parameter that the exit program must pass back to the API:

### Operation (CHGFP)

INPUT; CHAR(5)

The abbreviation for the operation being performed (CHGFP).

### File system job handle

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

The following parameters are the same as the parameters for the API.

### Open file handle

INPUT; CHAR(16)

### Move information

INPUT; CHAR(6)

### Distance to move

INPUT; BINARY(4)

### New offset

OUTPUT; BINARY(4) UNSIGNED

## API Functions

The QHFCHGFP API performs the standard functions described in “Standard HFS API Functions” on page 117.

## Exit Program Requirements

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118 and these additional functions:

- Checks for an attempt to set the file pointer to a negative position (that is, before the start of the file) or a position beyond the maximum value allowed in a 4-byte unsigned binary number, and signals an error if either occurs.
- Moves the file pointer the specified distance from the specified starting location, and records the file pointer’s new offset value.

## Error Messages for Exit Program Use

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F2D E	File pointer position not valid.
CPF1F2E E	Range of bytes in file in use.
CPF1F28 E	Damaged file.
CPF1F4E E	Move information value not valid.
CPF1F4F E	Distance to move value not valid.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F77 E	Severe parameter error occurred on call to file system.

---

## Exit Program for Close Directory (QHFCLODR) API

Required Parameter Group:	
1	Operation (CLODR)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	Open directory handle
<b>Input</b>	Char(16)
Optional Parameter Group:	
4	Close type
<b>Input</b>	Char(1)

Before applications can use the Close Directory (QHFCLODR) API with your file system, you must:

1. Write an exit program that performs the close directory operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Close Directory (QHFCLODR) API" on page 42.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFCLODR API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

### Required Parameter Group

The API passes this information to your exit program:

#### Operation (CLODR)

INPUT; CHAR(5)

The abbreviation for the operation being performed (CLODR).

#### File system job handle

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

The following parameter is the same as the parameter for the API.

#### Open directory handle

INPUT; CHAR(16)

### Optional Parameter Group

If your file system was registered with a Version 2 Release 3 Modification Level 0, this parameter is passed to your exit program for the Close Directory API.

## Close type

INPUT; CHAR(1)

The type of close operation to be performed. Valid values are:

0	If the exit program cannot close the directory, HFS does not mark the directory as closed.
1	Unconditional close. HFS marks the directory as closed regardless of what valid exception is returned by the exit program. HFS may call your exit program for an unconditional close at the end of the job and during the reclaim resource processing. If your file system is called to unconditionally close a directory, the directory should not be marked as open by the file system when control is returned to HFS.

HFS uses the close type value 0 when the QHFCLODR API is called to close the directory. The unconditional close type value 1 is used when:

- The job ends.
- Reclaim resource processing is done.
- The Deregister File System (QHFDGRFS) API is called, and the job is using the file system that is to be deregistered.

## API Functions

In addition to the standard functions described in “Standard HFS API Functions” on page 117, the QHFCLODR API performs these functions for your file system:

- Validates the open directory handle to ensure that the directory is open and the current user profile is the user that opened it.
- Passes the corresponding file system handle to the file system.

## Exit Program Requirements

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118 and one additional function. The exit program should close the directory, releasing the lock that the user obtained when the directory was opened, and invalidate the directory handle so that it cannot be used again.

## Error Messages for Exit Program Use

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F06 E	Directory in use.
CPF1F08 E	Damaged directory.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## Exit Program for Close Stream File (QHFCLOSF) API

Required Parameter Group:

- |              |                           |
|--------------|---------------------------|
| 1            | Operation (CLOSF)         |
| <b>Input</b> | Char(5)                   |
| 2            | File system job handle    |
| <b>Input</b> | Char(16)                  |
| 3            | Open file handle          |
| <b>Input</b> | Char(16)                  |
|              | Optional Parameter Group: |
| 4            | Close type                |
| <b>Input</b> | Char(1)                   |

Before applications can use the Close Stream File (QHFCLOSF) API with your file system, you must:

1. Write an exit program that performs the close stream file operation on behalf of the API. For a detailed description of the API and its calling parameters, see “Close Stream File (QHFCLOSF) API” on page 16.
2. Give the exit program’s name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFCLOSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

### Required Parameter Group

The API passes this information to your exit program:

#### Operation (CLOSF)

INPUT; CHAR(5)

The abbreviation for the operation being performed (CLOSF).

#### File system job handle

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

The following parameter is the same as the parameter for the API.

#### Open file handle

INPUT; CHAR(16)

### Optional Parameter Group

If your file system was registered with a Version 2 Release 3 Modification Level 0, this parameter is passed to your exit program for the Close Stream File API.

#### Close type

INPUT; CHAR(1)



The type of close operation to be performed. Valid values are:

0	If the exit program cannot close the file, HFS does not mark the file as closed.
1	Unconditional close. HFS marks the directory as closed regardless of what valid exception is returned by the exit program. HFS may call your exit program for an unconditional close at the end of the job and during the reclaim resource processing. If your file system is called to unconditionally close a file, the file should not be marked as open by the file system when control is returned to HFS.

HFS uses the close type value 0 when the QHFCLOSF API is called to close the file. The unconditional close type value 1 is used when:

- The job ends.
- Reclaim resource processing is done.
- The Deregister File System (QHFDGRFS) API is called, and the job is using the file system to be deregistered.

## API Functions

The QHFCLOSF API performs the standard functions described in “Standard HFS API Functions” on page 117.

## Exit Program Requirements

You must create an exit program that performs the standard functions described “Standard HFS Exit Program Requirements” on page 118 and these additional functions:

- Releases any byte locks that the job has on the file.
- Closes the file and invalidates the file system job handle so that the handle cannot be used again.

## Error Messages for Exit Program Use

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F06 E	Directory in use.
CPF1F28 E	Damaged file.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## Exit Program for Control File System (QHFCTLFS) API

Required Parameter Group:	
1	Operation (CTLFS)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	File handle
<b>Input</b>	Char(16)
4	Input data buffer
<b>Input</b>	Char(*)
5	Input data buffer length
<b>Input</b>	Binary(4)
6	Output data buffer
<b>Output</b>	Char(4)
7	Output data buffer length
<b>Input</b>	Binary(4)
8	Length of data returned
<b>Output</b>	Binary(4)

Before applications can use the Control File System (QHFCTLFS) API with your file system, you must:

1. Write an exit program that performs the control file system operation on behalf of the API. For a detailed description of the API and its calling parameters, see “Control File System (QHFCTLFS) API” on page 4.
2. Give the exit program’s name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFCTLFS API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

### Required Parameter Group

The following shows the input parameters that the API passes to your exit program and the output parameters that the exit program must pass back to the API:

#### Operation (CTLFS)

INPUT: CHAR(10)

The abbreviation for the operation being performed (CTLFS).

#### File system job handle

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

The following parameters are the same as the parameters for the API:

**File handle**  
INPUT; CHAR(16)

**Input data buffer**  
INPUT; CHAR(\*)

**Input data buffer length**  
INPUT; BINARY(4)

**Output data buffer**  
OUTPUT; CHAR(\*)

**Output data buffer length**  
INPUT; BINARY(4)

**Length of data returned**  
OUTPUT; BINARY(4)

## API Functions

The QHFCTLFS API performs the standard functions described in “Standard HFS API Functions” on page 117.

## Exit Program Requirements

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118.

## Error Messages for Exit Program Use

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F47 E	Buffer overflow occurred.
CPF1F53 E	Value for length of data buffer not valid.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## Exit Program for Copy Stream File (QHFCPYSF) API

Required Parameter Group:

1	Operation (CPYSF)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	Reserved
<b>Input</b>	Char(20)
4	Source file path name
<b>Input</b>	Char(*)
5	Source file path name length
<b>Input</b>	Binary(4)
6	Copy information
<b>Input</b>	Char(6)
7	Target file path name
<b>Input</b>	Char(*)
8	Target file path name length
<b>Input</b>	Binary(4)
9	File system names
<b>Input</b>	Char(20)

Before applications can use the Copy Stream File (QHFCPYSF) API with your file system, you must:

1. Write an exit program that performs the copy stream file operation on behalf of the API. For a detailed description of the API and its calling parameters, see “Copy Stream File (QHFCPYSF) API” on page 32.
2. Give the exit program’s name when you register the file system with the Register File System (QHFRGFS) API. In the registration-information parameter of the QHFRGFS API, indicate whether this exit program can be used for copy operations involving two different file systems.

After that, when an application calls the QHFCPYSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

### Required Parameter Group

The API passes this information to your exit program:

#### Operation (CPYSF)

INPUT; CHAR(5)

The abbreviation for the operation being performed (CPYSF).

#### File system job handle

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

**Reserved**

INPUT; CHAR(20)

Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

**Source file path name**

INPUT; CHAR(\*)

The API removes the file system name before passing the path name to the exit program.

**Source file path name length**

INPUT; BINARY(4)

**Copy information**

INPUT; CHAR(6)

**Target file path name**

INPUT; CHAR(\*)

The API removes the file system name before passing the path name to the exit program.

**Target file path name length**

INPUT; BINARY(4)

**File system names**

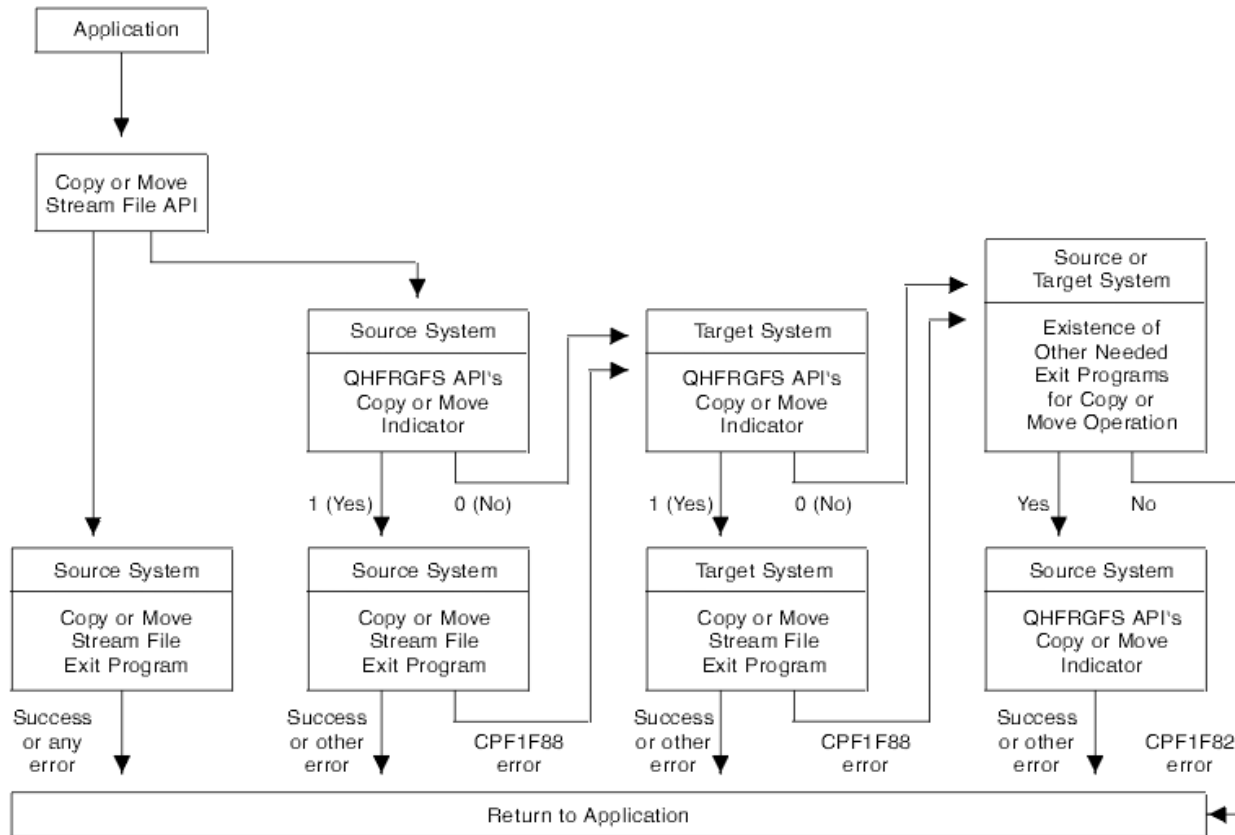
INPUT; CHAR(20)

This is not an API parameter. The API derives this information from its source and target file path name parameters. The first 10 characters contain the name of the source file system, and the second 10 characters contain the name of the target file system.

## API Functions

The QHFCPYSF API performs the standard functions described in “Standard HFS API Functions” on page 117.

When the source and target file systems are different, the API performs additional functions so that the file is copied by the most efficient means available. The following diagram outlines the processing steps. The steps are the same for copy and move operations. They are described in detail after the diagram.



After determining that the file systems differ, the API tries to perform the copy operation in several different ways. If a method fails, the API proceeds to the next method. The possible methods are to call the following exit programs in the sequence listed:

1. The source file system's Copy Stream File exit program
2. The target file system's Copy Stream File exit program
3. A series of other exit programs, such as those for opening, reading from, and writing to stream files, in the source and target file systems

The following paragraphs describe each method in detail.

**1. The source file system's Copy Stream File exit program:**

First, the API checks the information provided when the source file system was registered with the Register File System (QHFRGFS) API. The copy or move indicator is character 2 of the registration-information parameter described in the "Register File System (QHFRGFS) API" on page 54.

If the value of the source file system's copy or move indicator is 0 for no (indicating that this file system has no cross-file-system capability), the API proceeds to try the target file system.

If the value of the source file system's copy or move indicator is 1 for yes (indicating that this file system has some cross-file-system capability), the API calls the file system's Copy Stream File exit program.

If the exit program encounters an error in communicating with the source file system—for example, the exit program can work with some other file systems but not with this one—it returns message CPF1F88 to the API, which proceeds to the target file system.

If the exit program completes the copy or encounters any other type of error—for example, the exit program cannot find the file to be copied—it returns control to the API. The API resends any errors received from the exit program to the application and then returns control to the application.

**2. The target file system's Copy Stream File exit program:**

The API follows the same procedure as for the source file system, checking the copy or move indicator and then trying the target file system's Copy Stream File exit program. If the copy or move indicator is 0 for no or if the exit program returns message CPF1F88, the API proceeds to try the other exit programs.

**3. A series of other exit programs in the source and target file systems:**

If the source and target file systems' Copy Stream File exit programs have no cross-file-system capability at all, or if they have no such capability with respect to each other, the API might be able to perform a less efficient form of copy operation. If the following exit programs exist in the source and target file systems, the API tries to perform the copy operation. If any of these exit programs do not exist, the API returns message CPF1F82 and returns control to the application without performing the copy operation.

The required exit programs in the source file system are:

- Open Stream File (for the QHFOPNSF API)
- Read from Stream File (for the QHFRDSF API)
- Close Stream File (for the QHFCLOSF API)
- Retrieve Stream File Attributes (for the QHFRTVAT API)
- Delete Stream File (for the QHFDLTSF API). This exit program is required for both copy and move operations. However, it is used only during move operations.

The required exit programs in the target file system are:

- Open Stream File (for the QHFOPNSF API)
- Write to Stream File (for the QHFWRTSF API)
- Change File Pointer (for the QHFCHGFP API). This exit program is required for both copy and move operations. However, it is used only during copy operations in which the source file is being added to an existing target file.
- Close Stream File (for the QHFCLOSF API)
- Change Stream File Attributes (for the QHFCHGAT API)
- Delete Stream File (for the QHFDLTSF API)

The API uses the exit programs to perform their ordinary functions:

- The source file system's Open Stream File exit program opens the source file, and the target file system's Open Stream File exit program opens the target file.
- The source file system's Read from Stream File exit program and the target file system's Write to Stream File exit program repeatedly read from the source file and write to the target file until all the data has been copied from one to the other.
- The source file system's Retrieve Stream File Attributes exit program retrieves the attributes stored with the source file, and the target file system's Change Stream File Attributes exit program writes the attributes to the target file.
- The source and target file systems' Close Stream File exit programs close the source and target files.
- The source file system's Delete Stream File exit program deletes the source file after a successful move operation.

The target file system's Delete Stream File exit program is used during unsuccessful copy and move operations. If errors cause the failure of the operation as a whole, the exit program deletes any incomplete target files created during the operation. Target files that existed before the operation began are not deleted but might be partly changed.

## Exit Program Requirements

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118 and these additional functions:

- If the source and target paths to the files are the same, verifies that the source and target file names are different.
- If the source and target file systems are different, takes these actions:
  - Determines whether it is the source or the target file system by examining the exit program’s file-system-names parameter.
  - Performs the copy operation or returns control to the application or API as described in “API Functions” on page 69.

When the exit program has some cross-file-system capability but cannot complete this specific copy operation, it should return message CPF1F88 to the API. This tells the API that it might still be possible to perform the copy operation by other means. If unavoidable errors occur—for example, if the file being copied does not exist—then the exit program should return those errors to the API.

Whether or not the QHFCPYSF API calls this exit program for cross-file-system operations depends on information provided when this file system was registered. If the second character of the registration-information parameter of the Register File System (QHFRGFS) API has a value of 1 (yes), the QHFCPYSF API calls this exit program. If that character has a value of 0 (no), the QHFCPYSF API does not call this exit program for cross-file-system operations; instead, the API bypasses this exit program and proceeds to try the next available method of copying files.

- If the operation is replacing or adding a copy to a file, verifies that the target file exists.
- Ensures that the user has the required authority to both the source and target paths and files.
- Ensures that the user has read access to the source file and write access to the target file.

## Error Messages for Exit Program Use

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F21 E	File name not valid.
CPF1F22 E	File not found.
CPF1F23 E	New file name same as old file name.
CPF1F24 E	File name already exists.
CPF1F26 E	File in use.
CPF1F27 E	Authority not sufficient to access file.
CPF1F28 E	Damaged file.
CPF1F29 E	Use of reserved file name not allowed.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F48 E	Path name not valid.
CPF1F51 E	Copy information value not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F77 E	Severe parameter error occurred on call to file system.



Message ID	Error Message Text
CPF1F88 E	Unable to complete copy or move operation.

**Note:** You can use message CPF1F88 only when trying to perform cross-file-system copy operations. If you use it when copying files within a single file system, an Internal file system error message is returned to the application. You can use the exit program's file-system-names parameter to determine whether the move is across file systems.

Because this message does not always indicate an error, the application calling the QHFCPYSF API does not receive it. It is used only to communicate between the file system's exit program and the API.

Exit Program Introduced: V2R1

[Top](#) | ["Hierarchical File System APIs," on page 1](#) | [APIs by category](#)

---

## Exit Program for Create Directory (QHFCRTDR) API

Required Parameter Group:	
1	Operation (CRTDR)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	Reserved
<b>Input</b>	Char(20)
4	Path name
<b>Input</b>	Char(*)
5	Length of path name
<b>Input</b>	Char(*)
6	Attribute information table
<b>Input</b>	Char(*)
7	Length of attribute information table
<b>Input</b>	Binary(4)

Before applications can use the Create Directory (QHFCRTDR) API with your file system, you must:

1. Write an exit program that performs the create directory operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Create Directory (QHFCRTDR) API" on page 8.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFCRTDR API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

## Required Parameter Group

The API passes this information to your exit program:

### Operation (CRTDR)

INPUT; CHAR(5)

The abbreviation for the operation being performed (CRTDR).

### File system job handle

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

### Reserved

INPUT; CHAR(20)

Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

### Path name

INPUT; CHAR(\*)

The API removes the file system name before passing the path name to the exit program.

### Length of path name

INPUT; BINARY(4)

### Attribute information table

INPUT; CHAR(\*)

### Length of attribute information table

INPUT; BINARY(4)

## API Functions

The QHFCRTDR API performs the standard functions described in “Standard HFS API Functions” on page 117 and one additional function. The API verifies that the length of the attribute information table is not negative. It does not validate the attribute information in any way.

## Exit Program Requirements

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118 and these additional functions:

- Supports user-defined attribute types.
- Validates the attribute information.
- Associates the specified attributes with the directory when it is created.
- Returns an exception without creating a directory if another user has the higher-level directory locked in deny write mode. (The **higher-level directory** is the directory in which the new directory is being created.)

## Error Messages for Exit Program Use

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F04 E	Directory name already exists.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.

Message ID	Error Message Text
CPF1F08 E	Damaged directory.
CPF1F09 E	Use of reserved directory name not allowed.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F42 E	Attribute information table not valid.
CPF1F43 E	Attribute name not valid.
CPF1F44 E	Attribute value is not valid.
CPF1F46 E	Use of reserved attribute name not allowed.
CPF1F48 E	Path name not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

Top | "Hierarchical File System APIs," on page 1 | APIs by category

---

## Exit Program for Delete Directory (QHFDLTDR) API

Required Parameter Group:	
1	Operation (DLTDR)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	Reserved
<b>Input</b>	Char(20)
4	Path name
<b>Input</b>	Char(*)
5	Path name length
<b>Input</b>	Binary(4)

Before applications can use the Delete Directory (QHFDLTDR) API with your file system, you must:

1. Write an exit program that performs the delete directory operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Delete Directory (QHFDLTDR) API" on page 10.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFDLTDR API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

## Required Parameter Group

The API passes this information to your exit program:

### Operation (DLTDR)

INPUT; CHAR(5)

The abbreviation for the operation being performed (DLTDR).

### File system job handle

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

### Reserved

INPUT; CHAR(20)

Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

### Path name

INPUT; CHAR(\*)

The API removes the file system name before passing the path name to the exit program.

### Path name length

INPUT; BINARY(4)

## API Functions

The QHFDLTDR API performs the standard functions described in “Standard HFS API Functions” on page 117.

## Exit Program Requirements

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118 and these additional functions:

- Verifies that the directory being deleted is not in use.
- Deletes the directory and its associated directory entry.

## Error Messages for Exit Program Use

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F0A E	Delete directory operation not allowed.
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F48 E	Path name not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.

**Message ID**      **Error Message Text**  
CPF1F77 E          Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## Exit Program for Delete Stream File (QHFDLTSF) API

Required Parameter Group:

<b>1</b>	Operation (DLTSF)
<b>Input</b>	Char(5)
<b>2</b>	File system job handle
<b>Input</b>	Char(16)
<b>3</b>	Reserved
<b>Input</b>	Char(20)
<b>4</b>	Path name
<b>Input</b>	Char(*)
<b>5</b>	Path name length
<b>Input</b>	Binary(4)

Before applications can use the Delete Stream File (QHFDLTSF) API with your file system, you must:

1. Write an exit program that performs the delete stream file operation on behalf of the API. For a detailed description of the API and its calling parameters, see “Delete Stream File (QHFDLTSF) API” on page 35.
2. Give the exit program’s name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFDLTSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

### Required Parameter Group

The API passes this information to your exit program:

#### Operation (DLTSF)

INPUT; CHAR(5)

The abbreviation for the operation being performed (DLTSF).

#### File system job handle

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

#### Reserved

INPUT; CHAR(20)

Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

**Path name**

INPUT; CHAR(\*)

The API removes the file system name before passing the path name to the exit program.

**Path name length**

INPUT; BINARY(4)

## API Functions

The QHFDLTSF API performs the standard functions described in “Standard HFS API Functions” on page 117.

## Exit Program Requirements

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118 and these additional functions:

- Ensures that the file being deleted is not open or in use.
- Ensures that the file being deleted is not a read-only file.

## Error Messages for Exit Program Use

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F21 E	File name not valid.
CPF1F22 E	File not found.
CPF1F26 E	File in use.
CPF1F27 E	Authority not sufficient to access file.
CPF1F28 E	Damaged file.
CPF1F37 E	File is a read-only file.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F48 E	Path name not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## End Job Session Exit Program

Required Parameter Group:

- |              |                        |
|--------------|------------------------|
| 1            | Operation (TERM)       |
| <b>Input</b> | Char(5)                |
| 2            | File system job handle |
| <b>Input</b> | Char(16)               |

You must supply an End Job Session exit program for your file system. HFS support calls the End Job Session exit program whenever a job that uses the HFS APIs ends. The End Job Session exit program cleans up any work areas that the job used. If your Start Job Session exit program creates temporary work spaces, use the End Job Session exit program to delete them.

### Required Parameter Group

HFS support passes this information to the End Job Session exit program:

#### Operation (TERM)

INPUT; CHAR(5)

The abbreviation for the operation being performed (TERM, meaning end).

#### File system job handle

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

### Error Messages for Exit Program Use

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F76 E	Error occurred during end-job-session function

Exit Program Introduced: V2R1

[Top](#) | ["Hierarchical File System APIs," on page 1](#) | [APIs by category](#)

---

## Exit Program for Force Buffered Data (QHFFRCSF) API

Required Parameter Group:

- |              |                        |
|--------------|------------------------|
| 1            | Operation (FRCSF)      |
| <b>Input</b> | Char(5)                |
| 2            | File system job handle |
| <b>Input</b> | Char(16)               |
| 3            | Files to force         |
| <b>Input</b> | Char(16)               |

Before applications can use the Force Buffered Data (QHFFRCSF) API with your file system, you must:

1. Write an exit program that performs the force operation on behalf of the API. For a detailed description of the API and its calling parameters, see “Force Buffered Data (QHFFRCSF) API” on page 17.
2. Give the exit program’s name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFFRCSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

## Required Parameter Group

The API passes this information to your exit program:

### Operation (FRCSF)

INPUT; CHAR(5)

The abbreviation for the operation being performed (FRCSF).

### File system job handle

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

The following parameter is the same as the parameter for the API.

### Files to force

INPUT; CHAR(16)

## API Functions

The QHFFRCSF API performs the standard functions described in “Standard HFS API Functions” on page 117 and these additional functions:

- It verifies that the files to force parameter gives either a valid open file handle or hexadecimal zeros to indicate all files.
- If one file is specified, it calls the appropriate file system to force the data. Any exceptions received from the file system are either signaled or mapped into an error code, as the application requests.
- If all files are specified, it calls each file system used by the process once for each open file within that file system to force the data for that file. Any exceptions received from the file system are left in the job log, and the API continues processing the remaining open files until all are forced. Any exceptions are reported to the caller as a special error condition indicating that the attempt to force all files partially failed.

## Exit Program Requirements

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118.

## Error Messages for Exit Program Use

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F2B E	Write operation not allowed to file opened for read only.
CPF1F2E E	Range of bytes in file in use.
CPF1F28 E	Damaged file.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F61 E	No free space available on media.



Message ID	Error Message Text
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## Exit Program for Get Stream File Size (QHFGETSZ) API

Required Parameter Group:	
1	Operation (GETSZ)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	Open file handle
<b>Input</b>	Char(16)
4	File size
<b>Output</b>	Binary(4) Unsigned

Before applications can use the Get Stream File Size (QHFGETSZ) API with your file system, you must:

1. Write an exit program that performs the get size operation on behalf of the API. For a detailed description of the API and its calling parameters, see “Get Stream File Size (QHFGETSZ) API” on page 18.
2. Give the exit program’s name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFGETSZ API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

### Required Parameter Group

The following shows the input parameters that the API passes to your exit program and the output parameter that the exit program must pass back to the API:

#### Operation (GETSZ)

INPUT; CHAR(5)

The abbreviation for the operation being performed (GETSZ).

#### File system job handle

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

The following parameters are the same as the parameters for the API.

**Open file handle**

INPUT; CHAR(16)

**File size**

OUTPUT; BINARY(4) UNSIGNED

## API Functions

The QHFGETSZ API performs the standard functions described in “Standard HFS API Functions” on page 117.

## Exit Program Requirements

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118 and one additional function. The exit program should retrieve the size of the file as of the last write operation, excluding any object information stored with the file.

## Error Messages for Exit Program Use

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F28 E	Damaged file.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,”](#) on page 1 | [APIs by category](#)

---

## Exit Program for Lock and Unlock Range in Stream File (QHFLULSF) API

Required Parameter Group:	
1	Operation (LULSF)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	Open file handle
<b>Input</b>	Char(16)
4	Lock information
<b>Input</b>	Char(6)
5	File offset where lock begins
<b>Input</b>	Binary(4)
6	Bytes to lock
<b>Input</b>	Binary(4) Unsigned
7	File offset where unlock begins
<b>Input</b>	Binary(4) Unsigned
8	Bytes to unlock
<b>Input</b>	Binary(4) Unsigned

Before applications can use the Lock and Unlock Range in Stream File (QHFLULSF) API with your file system, you must:

1. Write an exit program that performs the lock and unlock operation on behalf of the API. For a detailed description of the API and its calling parameters, see “Lock and Unlock Range in Stream File (QHFLULSF) API” on page 19.
2. Give the exit program’s name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFLULSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

### Required Parameter Group

The API passes this information to your exit program:

#### Operation (LULSF)

INPUT; CHAR(5)

The abbreviation for the operation being performed (LULSF).

#### File system job handle

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

The following parameters are the same as the parameters for the API.

**Open file handle**

INPUT; CHAR(16)

**Lock information**

INPUT; CHAR(6)

**File offset where lock begins**

INPUT; BINARY(4) UNSIGNED

**Bytes to lock**

INPUT; BINARY(4) UNSIGNED

**File offset where unlock begins**

INPUT; BINARY(4) UNSIGNED

**Bytes to unlock**

INPUT; BINARY(4) UNSIGNED

## API Functions

The QHFLULSF API performs the standard functions described in “Standard HFS API Functions” on page 117 and these additional functions:

- Verifies that the bytes to lock and bytes to unlock parameters are not both zero. In other words, the API ensures that a lock, unlock, or both operations are to be performed. If both parameters are zero, an error is returned.
- Verifies that the lock information parameter contains valid values, and that the lock mode value is applicable to the lock or unlock operation requested.

## Exit Program Requirements

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118 and these additional functions:

- Verifies that the file offsets to lock and unlock are valid offsets.
- Verifies that the number of bytes to lock and unlock are valid values.
- For an unlock operation, verifies that this job previously locked the range.
- For a lock operation, verifies that no part of the range already has a lock that does not allow the access requested.
- Unlocks or locks the range requested.

## Error Messages for Exit Program Use

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F2E E	Range of bytes in file in use.
CPF1F2F E	Unlock range of bytes in file failed.
CPF1F28 E	Damaged file.
CPF1F32 E	Number of locks on file exceeds limit.
CPF1F4B E	Value for number of bytes not valid.
CPF1F4C E	Lock information value not valid.
CPF1F4D E	File offset value not valid.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.

**Message ID**      **Error Message Text**  
CPF1F77 E      Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## Exit Program for Move Stream File (QHFMVSF) API

Required Parameter Group:	
1	Operation (MOVSF)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	Reserved
<b>Input</b>	Char(20)
4	Source file path name
<b>Input</b>	Char(*)
5	Source file path name length
<b>Input</b>	Binary(4)
6	Target file path name
<b>Input</b>	Char(*)
7	Target file path name length
<b>Input</b>	Binary(4)
8	File system names
<b>Input</b>	Char(20)

Before applications can use the Move Stream File (QHFMVSF) API with your file system, you must:

1. Write an exit program that performs the move stream file operation on behalf of the API. For a detailed description of the API and its calling parameters, see “Move Stream File (QHFMVSF) API” on page 36.
2. Give the exit program’s name when you register the file system with the Register File System (QHFRGFS) API. In the registration-information parameter of the QHFRGFS API, indicate whether this exit program can be used for move operations involving two different file systems.

After that, when an application calls the QHFMVSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

### Required Parameter Group

The API passes this information to your exit program:

**Operation (MOVSF)**  
INPUT; CHAR(5)

The abbreviation for the operation being performed (MOVSF).

**File system job handle**

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

**Reserved**

INPUT; CHAR(20)

Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

**Source file path name**

INPUT; CHAR(\*)

The API removes the file system name before passing the path name to the exit program.

**Source file path name length**

INPUT; BINARY(4)

**Target file path name**

INPUT; CHAR(\*)

The API removes the file system name before passing the path name to the exit program.

**Target file path name length**

INPUT; BINARY(4)

**File system names**

INPUT; CHAR(20)

This is not an API parameter. The API derives this information from its source and target file path name parameters. The first 10 characters contain the name of the source file system, and the second 10 characters contain the name of the target file system.

## API Functions

The QHFMOVSF API performs the standard functions described in “Standard HFS API Functions” on page 117.

When the source and target file systems are different, the API performs additional functions so that the file is moved by the most efficient means available. The processing steps are the same as those described for the Copy Stream File exit program in “API Functions,” with these exceptions:

- The Move Stream File (QHFMOVSF) API and Move Stream File exit program are used instead of the Copy Stream File API and exit program.
- After successful completion of the move operation, the source file system’s Delete Stream File exit program is used to delete the source file.

## Exit Program Requirements

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118 and these additional functions:

- When the source and target file systems are different, performs the same actions described for the Copy Stream File exit program; see “Exit Program Requirements.”
- Verifies that the target file does not exist.
- Ensures that the file being moved is not open or in use.

## Error Messages for Exit Program Use

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F03 E	New directory name same as old directory name.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F21 E	File name not valid.
CPF1F22 E	File not found.
CPF1F24 E	File name already exists.
CPF1F26 E	File in use.
CPF1F27 E	Authority not sufficient to access file.
CPF1F28 E	Damaged file.
CPF1F29 E	Use of reserved file name not allowed.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F48 E	Path name not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F77 E	Severe parameter error occurred on call to file system.
CPF1F88 E	Unable to complete copy or move operation.

**Note:** You can use message CPF1F88 only when trying to perform cross-file-system move operations. If you use it when moving files within a single file system, an Internal file system error message is returned to the application. You can use the exit program's file-system-names parameter to determine whether the move is across file systems.

Because this message does not always indicate an error, the application calling the QHFMOVSF API does not receive it. It is used only to communicate between the file system's exit program and the API.

Exit Program Introduced: V2R1

[Top](#) | ["Hierarchical File System APIs," on page 1](#) | [APIs by category](#)

---

## Exit Program for Open Directory (QHFOPNDR) API

Required Parameter Group:

1	Operation (OPNDR)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	Reserved
<b>Input</b>	Char(20)
4	Open directory handle
<b>Output</b>	Char(16)
5	Path name
<b>Input</b>	Char(*)
6	Path name length
<b>Input</b>	Binary(4)
7	Open information
<b>Input</b>	Char(6)
8	Attribute selection table
<b>Input</b>	Char(4)
9	Length of attribute selection table
<b>Input</b>	Binary(4)

Before applications can use the Open Directory (QHFOPNDR) API with your file system, you must:

1. Write an exit program that performs the open directory operation on behalf of the API. For a detailed description of the API and its calling parameters, see “Open Directory (QHFOPNDR) API” on page 43.
2. Give the exit program’s name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFOPNDR API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

### Required Parameter Group

The following shows the input parameters that the API passes to your exit program and the output parameter that the exit program must pass back to the API:

#### Operation (OPNDR)

INPUT; CHAR(5)

The abbreviation for the operation being performed (OPNDR).

#### File system job handle

INPUT; CHAR(16)

The work area or job identifier used by the file system.



**Reserved**

INPUT; CHAR(20)

Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

**Open directory handle**

OUTPUT; CHAR(16)

**Path name**

INPUT; CHAR(\*)

The API removes the file system name before passing the path name to the exit program.

**Path name length**

INPUT; BINARY(4)

**Open information**

INPUT; CHAR(6)

The exit program can ignore character 2, which describes the type of open operation to perform. This field is used by HFS support during job cleanup if the job ends before the file is closed.

**Attribute selection table**

INPUT; CHAR(\*)

**Length of the attribute selection table**

INPUT; BINARY(4)

## API Functions

The QHFOPNDR API performs the standard functions described in “Standard HFS API Functions” on page 117. The API does not validate the attribute selection table in any way.

## Exit Program Requirements

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118 and these additional functions:

- Verifies that all directories in the path exist.
- If the last element of the path name is a specific name, opens that specific directory.
- If the last element of the path name is a generic name:
  - Opens the directory specified as the next-to-last element.
  - Interprets the generic name so that only directory entries that match it are available for subsequent read operations.
- Validates the attribute selection table.
- Locks the directory and its attributes according to the specified lock mode.
- Associates the attribute selection table with the open directory so that subsequent read operations using the QHFRDDR API return only the attributes selected when the directory was opened.

## Error Messages for Exit Program Use

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.

Message ID	Error Message Text
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F43 E	Attribute name not valid.
CPF1F45 E	Attribute selection table not valid.
CPF1F48 E	Path name not valid.
CPF1F49 E	Open information value not valid.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

[Top](#) | ["Hierarchical File System APIs," on page 1](#) | [APIs by category](#)

---

## Exit Program for Open Stream File (QHFOPNSF) API

Required Parameter Group:	
1	Operation (OPNSF)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	Reserved
<b>Input</b>	Char(20)
4	Open file handle
<b>Output</b>	Char(16)
5	Path name
<b>Input</b>	Char(*)
6	Path name length
<b>Input</b>	Binary(4)
7	Open information
<b>Input</b>	Char(10)
8	Attribute information table
<b>Input</b>	Char(*)
9	Length of attribute information table
<b>Input</b>	Binary(4)
10	Action taken
<b>Output</b>	Char(1)

Before applications can use the Open Stream File (QHFOPNSF) API with your file system, you must:

1. Write an exit program that performs the open stream file operation on behalf of the API. For a detailed description of the API and its calling parameters, see “Open Stream File (QHFOFNSF) API” on page 22.
2. Give the exit program’s name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFOFNSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

## Required Parameter Group

The following shows the input parameters that the API passes to your exit program and the output parameters that the exit program must pass back to the API:

### Operation (OPNSF)

INPUT; CHAR(5)

The abbreviation for the operation being performed (OPNSF).

### File system job handle

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

### Reserved

INPUT; CHAR(20)

Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

### Open file handle

OUTPUT; CHAR(16)

### Path name

INPUT; CHAR(\*)

The API removes the file system name before passing the path name to the exit program.

### Path name length

INPUT; BINARY(4)

### Open information

INPUT; CHAR(10)

The exit program can ignore character 7, which describes the type of open operation to perform. This field is used by OS/400 HFS support during job cleanup if the job ends before the file is closed.

### Attribute information table

INPUT; CHAR(\*)

### Length of the attribute information table

INPUT; BINARY(4)

### Action taken

OUTPUT; CHAR(1)

## API Functions

The QHFOFNSF API performs the standard functions described in “Standard HFS API Functions” on page 117 and these additional functions:

- Verifies that the length of the attribute information table is not negative.

- Ensures that the file is closed during normal job cleanup, in case the user forgets to close it before ending the job.
- Handles the type-of-open value represented by character 7 of the open information parameter. The file system does not need to take any action on the basis of this value.

## Exit Program Requirements

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118 and these additional functions:

- Checks for previous open operations that have lock modes conflicting with the requested access mode for this file.
- Attempts to take the action designated by the open information. The action can be opening an existing file, opening and replacing an existing file, or creating and opening a new file.
- If the action is successful, assigns a file handle, locks the file and its attributes, and returns the handle and action taken to the API. The API does not return this handle to the application. The API creates a handle of its own to return to the application. This procedure improves API performance and ensures that handles are unique across file systems.

## Error Messages for Exit Program Use

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F2A E	Number of open files exceeds limit.
CPF1F21 E	File name not valid.
CPF1F22 E	File not found.
CPF1F24 E	File name already exists.
CPF1F26 E	File in use.
CPF1F27 E	Authority not sufficient to access file.
CPF1F28 E	Damaged file.
CPF1F29 E	Use of reserved file name not allowed.
CPF1F37 E	File is a read-only file.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F42 E	Attribute information table not valid.
CPF1F43 E	Attribute name not valid.
CPF1F44 E	Attribute value is not valid.
CPF1F46 E	Use of reserved attribute name not allowed.
CPF1F48 E	Path name not valid.
CPF1F49 E	Open information value not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

## Exit Program for Read Directory Entries (QHFRDDR) API

Required Parameter Group:	
1	Operation (RDDR)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	Open directory handle
<b>Input</b>	Char(16)
4	Data buffer
<b>Output</b>	Char(*)
5	Data buffer length
<b>Input</b>	Binary(4)
6	Number of directory entries to read
<b>Input</b>	Binary(4)
7	Number of directory entries read
<b>Output</b>	Binary(4)
8	Length of data returned
<b>Output</b>	Binary(4)

Before applications can use the Read Directory Entries (QHFRDDR) API with your file system, you must:

1. Write an exit program that performs the read directory entries operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Read Directory Entries (QHFRDDR) API" on page 46.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFRDDR API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

### Required Parameter Group

The following shows the input parameters that the API passes to your exit program and the output parameter that the exit program must pass back to the API:

#### Operation (RDDR)

INPUT; CHAR(5)

The abbreviation for the operation being performed (RDDR).

#### File system job handle

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

The following parameters are the same as the parameters for the API.

**Open directory handle**

INPUT; CHAR(16)

**Data buffer**

OUTPUT; CHAR(\*)

**Data buffer length**

INPUT; BINARY(4)

**Number of directory entries to read**

INPUT; BINARY(4)

**Number of directory entries read**

OUTPUT; BINARY(4)

**Length of data returned**

OUTPUT; BINARY(4)

## API Functions

The QHFRDDR API performs the standard functions described in “Standard HFS API Functions” on page 117 and one additional function. The API validates the open directory handle to ensure that the directory is open and the current user profile is the user that opened it.

## Exit Program Requirements

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118 and these additional functions:

- Retrieves the directory entry information. The file system should return only attributes selected with the attribute selection table when the directory was opened. The file system must build the table and set the number of directory entries actually read and the length of data returned.
- If a requested attribute is not associated with a directory entry, returns the attribute name and the length of the attribute value, which is zero.
- Increases the directory pointer value to reflect its new position after directory entries are read.

In addition, your file system’s documentation should describe the order in which directory entries are returned (for example, alphabetic or last-used date).

## Error Messages for Exit Program Use

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F08 E	Damaged directory.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F47 E	Buffer overflow occurred.
CPF1F53 E	Value for length of data buffer not valid.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

## Exit Program for Read from Stream File (QHFRDSF) API

Required Parameter Group:	
1	Operation (RDSF)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	Open file handle
<b>Input</b>	Char(16)
4	Data buffer
<b>Output</b>	Char(*)
5	Bytes to read
<b>Input</b>	Binary(4)
6	Bytes actually read
<b>Output</b>	Binary(4)

Before applications can use the Read from Stream File (QHFRDSF) API with your file system, you must:

1. Write an exit program that performs the read operation on behalf of the API. For a detailed description of the API and its calling parameters, see Read from Stream File (QHFRDSF) API.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFRDSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

### Required Parameter Group

The following shows the input parameters that the API passes to your exit program and the output parameters that the exit program must pass back to the API:

#### Operation (RDSF)

INPUT; CHAR(5)

The abbreviation for the operation being performed (RDSF).

#### File system job handle

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

The following parameters are the same as the parameters for the API.

#### Open file handle

INPUT; CHAR(16)

**Data buffer**

OUTPUT; CHAR(\*)

**Bytes to read**

INPUT; BINARY(4)

**Bytes actually read**

OUTPUT; BINARY(4)

## API Functions

The QHFRDSF API performs the standard functions described in “Standard HFS API Functions” on page 117.

## Exit Program Requirements

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118 and these additional functions:

- Verifies that the file was previously opened with an access mode of read only or read/write.
- Checks the range of bytes being read to make sure no part of the range is locked in deny read/write mode, which would preclude this operation.
- Reads the number of bytes specified from the file, starting at the current file pointer position, and places the data read in the data buffer.
- Records the number of bytes actually read. If the end of the file is reached during the read operation, this number is less than the number specified in the bytes-to-read parameter.
- Increases the value of the file pointer by the number of bytes read.
- If the read operation is not successful, sets the bytes actually read to zero and returns an exception describing the error to the API.

## Error Messages for Exit Program Use

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F2C E	Read operation not allowed to file opened for write only.
CPF1F2E E	Range of bytes in file in use.
CPF1F28 E	Damaged file.
CPF1F35 E	Read file operation failed.
CPF1F4B E	Value for number of bytes not valid.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)



---

## Exit Program for Rename Directory (QHFRNMDR) API

Required Parameter Group:	
1	Operation (RNMDR)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	Reserved
<b>Input</b>	Char(20)
4	Path name
<b>Input</b>	Char(*)
5	Path name length
<b>Input</b>	Binary(4)
6	New directory name
<b>Input</b>	Char(*)
7	New directory name length
<b>Input</b>	Binary(4)

Before applications can use the Rename Directory (QHFRNMDR) API with your file system, you must:

1. Write an exit program that performs the rename directory operation on behalf of the API. For a detailed description of the API and its calling parameters, see “Rename Directory (QHFRNMDR) API” on page 11.
2. Give the exit program’s name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFRNMDR API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

### Required Parameter Group

The API passes this information to your exit program:

#### Operation (RNMDR)

INPUT; CHAR(5)

The abbreviation for the operation being performed (RNMDR).

#### File system job handle

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

#### Reserved

INPUT; CHAR(20)

Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

**Path name**

INPUT; CHAR(\*)

The API removes the file system name before passing the path name to the exit program.

**Path name length**

INPUT; BINARY(4)

**New directory name**

INPUT; CHAR(\*)

**New directory name length**

INPUT; BINARY(4)

**API Functions**

The QHFRNMDR API performs the standard functions described in “Standard HFS API Functions” on page 117.

**Exit Program Requirements**

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118 and these additional functions:

- Verifies that the new directory does not already exist and that it has a different name from the old directory.
- Verifies that the directory is not in use before renaming it.
- Renames the directory.

**Error Messages for Exit Program Use**

This section lists the messages that the exit program can return to the API.

<b>Message ID</b>	<b>Error Message Text</b>
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F03 E	New directory name same as old directory name.
CPF1F04 E	Directory name already exists.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F09 E	Use of reserved directory name not allowed.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F48 E	Path name not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## Exit Program for Rename Stream File (QHFRNMSF) API

Required Parameter Group:	
1	Operation (RNMSF)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	Reserved
<b>Input</b>	Char(20)
4	Path name
<b>Input</b>	Char(*)
5	Path name length
<b>Input</b>	Binary(4)
6	New file name
<b>Input</b>	Char(*)
7	New file name length
<b>Input</b>	Binary(4)

Before applications can use the Rename Stream File (QHFRNMSF) API with your file system, you must:

1. Write an exit program that performs the rename stream file operation on behalf of the API. For a detailed description of the API and its calling parameters, see “Rename Stream File (QHFRNMSF) API” on page 38.
2. Give the exit program’s name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFRNMSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

### Required Parameter Group

The API passes this information to your exit program:

#### Operation (RNMSF)

INPUT; CHAR(5)

The abbreviation for the operation being performed (RNMSF).

#### File system job handle

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

#### Reserved

INPUT; CHAR(20)

Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

**Path name**

INPUT; CHAR(\*)

The API removes the file system name before passing the path name to the exit program.

**Path name length**

INPUT; BINARY(4)

**New file name**

INPUT; CHAR(\*)

**New file name length**

INPUT; BINARY(4)

**API Functions**

The QHFRNMSF API performs the standard functions described in “Standard HFS API Functions” on page 117.

**Exit Program Requirements**

You must create an exit program that performs the standard functions described “Standard HFS Exit Program Requirements” on page 118 and these additional functions:

- Verifies that the new file name does not exist and is different from the current file name.
- Ensures that the file being renamed is not open or in use.

**Error Messages for Exit Program Use**

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F21 E	File name not valid.
CPF1F22 E	File not found.
CPF1F23 E	New file name same as old file name.
CPF1F24 E	File name already exists.
CPF1F26 E	File in use.
CPF1F27 E	Authority not sufficient to access file.
CPF1F28 E	Damaged file.
CPF1F29 E	Use of reserved file name not allowed.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F48 E	Path name not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

Top | “Hierarchical File System APIs,” on page 1 | APIs by category

---

## Exit Program for Retrieve Directory Entry Attributes (QHFRTVAT) API

Required Parameter Group:	
1	Operation (RTVAT)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	Reserved
<b>Input</b>	Char(20)
4	Path name
<b>Input</b>	Char(*)
5	Path name length
<b>Input</b>	Binary(4)
6	Attribute selection table
<b>Input</b>	Char(*)
7	Length of attribute selection table
<b>Input</b>	Binary(4)
8	Attribute information table
<b>Output</b>	Char(*)
9	Length of attribute information table
<b>Input</b>	Binary(4)
10	Length of data returned
<b>Output</b>	Binary(4)

Before applications can use the Retrieve Directory Entry Attributes (QHFRTVAT) API with your file system, you must:

1. Write an exit program that performs the retrieve attributes operation on behalf of the API. For a detailed description of the API and its calling parameters, see “Retrieve Directory Entry Attributes (QHFRTVAT) API” on page 49.
2. Give the exit program’s name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFRTVAT API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

### Required Parameter Group

The following shows the input parameters that the API passes to your exit program and the output parameters that the exit program must pass back to the API:

#### Operation (RTVAT)

INPUT; CHAR(5)

The abbreviation for the operation being performed (RTVAT).

**File system job handle**

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

**Reserved**

INPUT; CHAR(20)

Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

**Path name**

INPUT; CHAR(\*)

The API removes the file system name before passing the path name to the exit program.

**Path name length**

INPUT; BINARY(4)

**Attribute selection table**

INPUT; CHAR(\*)

**Length of the attribute selection table**

INPUT; BINARY(4)

**Attribute information table**

OUTPUT; CHAR(\*)

**Length of the attribute information table**

INPUT; BINARY(4)

**Length of data returned**

OUTPUT; BINARY(4)

**API Functions**

The QHFRTVAT API performs the standard functions described in “Standard HFS API Functions” on page 117. The API does not validate the attribute selection table, the attribute information table, or the length of the attribute information table.

**Exit Program Requirements**

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118 and these additional functions:

- Validates the attribute selection table.
- Builds the attribute information table to return the requested attributes to the application.

**Error Messages for Exit Program Use**

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F21 E	File name not valid.
CPF1F22 E	File not found.
CPF1F26 E	File in use.
CPF1F27 E	Authority not sufficient to access file.

Message ID	Error Message Text
CPF1F28 E	Damaged file.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F42 E	Attribute information table not valid.
CPF1F43 E	Attribute name not valid.
CPF1F45 E	Attribute selection table not valid.
CPF1F47 E	Buffer overflow occurred.
CPF1F48 E	Path name not valid.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

Top | "Hierarchical File System APIs," on page 1 | APIs by category

---

## Exit Program for Set Stream File Size (QHFSETSZ) API

Required Parameter Group:	
1	Operation (SETSZ)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	Open file handle
<b>Input</b>	Char(16)
4	File size
<b>Input</b>	Binary(4) Unsigned

Before applications can use the Set Stream File Size (QHFSETSZ) API with your file system, you must:

1. Write an exit program that performs the set size operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Set Stream File Size (QHFSETSZ) API" on page 29.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFSETSZ API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

### Required Parameter Group

The API passes this information to your exit program:

**Operation (SETSZ)**  
INPUT; CHAR(5)

The abbreviation for the operation being performed (SETSZ).

#### **File system job handle**

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

The following parameters are the same as the parameters for the API.

#### **Open file handle**

INPUT; CHAR(16)

#### **File size**

INPUT; BINARY(4) UNSIGNED

## **API Functions**

The QHFSETSZ API performs the standard functions described in “Standard HFS API Functions” on page 117.

## **Exit Program Requirements**

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118 and these additional functions:

- Checks for byte locks that conflict with changing the size of the file, and returns an exception if any are found. The application cannot set the file size into or beyond a locked range.
- Verifies that the file size parameter value is valid for that file system.
- Increases or decreases the file size.

## **Error Messages for Exit Program Use**

This section lists the messages that the exit program can return to the API.

<b>Message ID</b>	<b>Error Message Text</b>
CPF1F2B E	Write operation not allowed to file opened for read only.
CPF1F2E E	Range of bytes in file in use.
CPF1F28 E	Damaged file.
CPF1F4B E	Value for number of bytes not valid.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)



---

## Start Job Session Exit Program

Required Parameter Group:

1	Operation (INIT)
<b>Input</b>	Char(5)
2	File system job handle
<b>Output</b>	Char(16)
3	File system name
<b>Input</b>	Char(10)

Before applications can use the HFS APIs with your file system, you must supply a Start Job Session exit program for the file system.

The Start Job Session exit program controls access to the file system as a whole for each job in which that file system is used. The first time an application refers to a specific file system within a job by calling any HFS API, the HFS API performs these operations before performing its own function:

1. Checks the application's authority to the Start Job Session exit program. Authority to the Start Job Session exit program provides authority to all other exit programs that are registered for use with the file system.
2. Locks the Start Job Session exit program in shared read (\*SHRRD) mode to keep the file system from being deregistered while in use.
3. Calls the Start Job Session exit program. The Start Job Session exit program sets up a 16-byte area called a job handle for the file system to use during the job, and returns the job handle to the HFS API. The file system can use the job handle to store a pointer to a separate work area or as a work area in itself.

### Required Parameter Group

The following shows the input parameters that the API passes to your exit program and the output parameter that the exit program must pass back to the API:

#### Operation (INIT )

INPUT; CHAR(5)

The abbreviation for the operation being performed (the letters INIT, for initialize, followed by a blank).

#### File system job handle

OUTPUT; CHAR(16)

The work area or job handle for use by the file system. The file system returns the job handle to HFS on the Start Job Session. On all subsequent API calls, HFS returns the job handle to the file system.

The file system can keep whatever you choose in the job handle. For example, the job handle might contain a pointer giving the address of another work area or a control block used by the file system.

#### File system name

INPUT; CHAR(10)

The name of the file system received from the application in the call to the HFS API. This parameter specifies which name the file system should use when it issues exceptions.

## Error Messages for Exit Program Use

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F75 E	Error occurred during start-job-session function.

Exit Program Introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## Exit Program for Write to Stream File (QHFWRTSF) API

Required Parameter Group:

1	Operation (WRTSF)
<b>Input</b>	Char(5)
2	File system job handle
<b>Input</b>	Char(16)
3	Open file handle
<b>Input</b>	Char(16)
4	Data buffer
<b>Input</b>	Char(*)
5	Bytes to write
<b>Input</b>	Binary(4)
6	Bytes actually written
<b>Output</b>	Binary(4)

Before applications can use the Write to Stream File (QHFWRTSF) API with your file system, you must:

1. Write an exit program that performs the write operation on behalf of the API. For a detailed description of the API and its calling parameters, see “Write to Stream File (QHFWRTSF) API” on page 30.
2. Give the exit program’s name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFWRTSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

### Required Parameter Group

The following shows the input parameters that the API passes to your exit program and the output parameters that the exit program must pass back to the API:

**Operation (WRTSF)**  
INPUT; CHAR(5)

The abbreviation for the operation being performed (WRTSF).

**File system job handle**

INPUT; CHAR(16)

The work area or job identifier for use by the file system.

The following parameters are the same as the parameters for the API.

**Open file handle**

INPUT; CHAR(16)

**Data buffer**

INPUT; CHAR(\*)

**Bytes to write**

INPUT; BINARY(4)

**Bytes actually written**

OUTPUT; BINARY(4)

**API Functions**

The QHFWRTSF API performs the standard functions described in “Standard HFS API Functions” on page 117.

**Exit Program Requirements**

You must create an exit program that performs the standard functions described in “Standard HFS Exit Program Requirements” on page 118 and these additional functions:

- Verifies that the file was previously opened with an access mode of write or read/write.
- Makes sure that no part of the range of bytes being written is locked in a way that denies access to this operation.
- Writes the number of bytes specified in the data buffer to the file, starting at the current file pointer position.
- Records the number of bytes actually written. Unless an error occurs, this number must be the same as the number specified in the bytes-to-write parameter.
- Increases the value of the file pointer by the number of bytes written.
- If the write operation is not successful, sets the bytes actually written to zero and signals an exception describing the error to the API.

**Error Messages for Exit Program Use**

This section lists the messages that the exit program can return to the API.

Message ID	Error Message Text
CPF1F2B E	Write operation not allowed to file opened for read only.
CPF1F2E E	Range of bytes in file in use.
CPF1F28 E	Damaged file.
CPF1F34 E	Attempted write operation beyond file size limit.
CPF1F36 E	Write file operation failed.
CPF1F4B E	Value for number of bytes not valid.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.

**Message ID**      **Error Message Text**  
CPF1F77 E      Severe parameter error occurred on call to file system.

Exit Program Introduced: V2R1

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## Concepts

These are the concepts for this category.

---

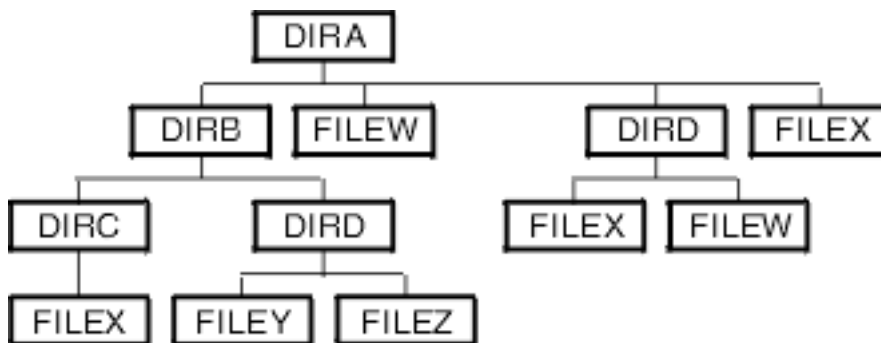
## HFS Concepts

The HFS APIs work with units of information or objects that belong to existing, hierarchical file systems. A **file system** is the operating system’s method of controlling the format of information on storage media and performing input and output operations to the objects that contain the information. The document library services (DLS) file system is one example of a file system.

The basic units of nonrelational information in a file system are usually called **files**. Files are sometimes called **byte-stream files** or **stream files** because they consist of a stream of bytes with no specific record structure.

A **hierarchical file system** arranges information units in a multilevel, tree-like structure, as IBM<sup>(R)</sup> DOS does. Files are grouped into larger units usually called directories. A **directory** can contain both files and subordinate directories. A directory contains no data of its own but is simply a named group of files and other directories. Two objects with the same name cannot exist in the same directory, but one directory can contain directories or files with the same names as those in another directory.

**HFS Directory—Scenario:** The following diagram illustrates the structure of a hierarchical directory:



In the preceding diagram, the topmost directory, DIRA, contains both directories, DIRB and DIRD, and files, FILEW and FILEX. Directory DIRD contains only files. Directory DIRB contains only directories. In this structure, there are two directories named DIRD, one in directory DIRA and one in directory DIRA/DIRB. There are also three files named FILEX, one in directory DIRA, one in DIRA/DIRD, and one in DIRA/DIRB/DIRC.

The specific location and name of a file or directory are represented in a multipart name called a path name. A **path name** starts with a slash (/) and consists of elements separated by slashes. The first element of the path name is the name of the file system. The remaining elements specify the applicable directory and file names; the last element can be either a file or a directory, but the rest must be directories.

**HFS Path Name—Example:** The path name /QDLS/DIRA/FILEW refers to file FILEW in directory DIRA in file system QDLS.

**Optical Path Name—Example:** The path name /QOPT/VOL1/DIR1/FILEA refers to file FILEA in directory DIR1 on volume VOL1 in file system QOPT.

**Note:** The optical file system requires an optical volume name following the file system name in the path.

Top | “Hierarchical File System APIs,” on page 1 | APIs by category

---

## HFS Use—Requirements

Before you can use an HFS API to work with a particular file system, these conditions must be met:

- You must have \*USE authority to the API. This gives you the authority to call the API from your programs.
- You must have authority to use the file system. Authority to the file system is controlled by the file system’s job startup program, described in “Start Job Session Exit Program” on page 105. If you have authority to use the file system’s Start Job exit program, you have authority to the file system as a whole.

Authority to use files and directories within a file system is controlled by the file system itself.

- The HFS API you want to use must be available for use with the file system. Some file systems might not support all the HFS APIs described in these articles.

“Hierarchical File System APIs,” on page 1 | APIs by category

---

## HFS Directory Entry Attributes

Every file and directory in a file system has a corresponding directory entry. The **directory entry** is created automatically by the file system when the stream file is opened or the directory is created. It is stored in the directory in which the file or directory is located and contains descriptive information, such as the item’s creation date and whether it is a file or directory. The items of information are called **directory entry attributes** or simply **attributes**. Each attribute has a name and a value.

### Standard HFS Directory Entry Attributes

Some directory entry attributes are created automatically when the directory entry is created. These attributes are called **standard attributes**. Their names start with the letter Q so you can identify them easily. Each file system determines which standard attributes you can specify when working with directory entries.

The standard attributes for directory entries are:

#### QNAME

CHAR(\*)

The current name of a file or directory.

Do not specify this attribute in the attribute information table or attribute selection table used with some APIs. The name is either already specified or disallowed, as follows:

- When you create a directory (or open a stream file) and when you retrieve directory entry attributes, the object’s name is already specified in the API’s path name parameter.
- The Read Directory Entries (QHFRDDR) API always returns the QNAME attribute to identify the name of the directory entry.

- You cannot use the Change Directory Entry Attributes (QHFCHGAT) API to change this attribute and rename the object. You must use the Rename Directory (QHFRNMDR) or Rename Stream File (QHFRNMSF) API to rename the object.

#### **QFILSIZE**

BINARY(4) UNSIGNED

The size of a file's data, in bytes.

This attribute applies only to files. Thus, for directories, it has a value of zero. If you specify it for a directory, it is ignored.

#### **QALCSIZE**

BINARY(4) UNSIGNED

For a file, the allocated size of the file in bytes. The allocated size is the amount of space the file system actually uses to store the file.

This attribute applies only to files. Thus, for directories, it has a value of zero. If you specify it for a directory, it is ignored.

#### **QCRTDTTM**

CHAR(13)

The date and time of day the file or directory was created, in CYYMMDDHHMMSS format.

You cannot specify this attribute when creating a directory or file. However, you can specify it on calls to the Retrieve Directory Entry Attributes (QHFRTVAT) and Change Directory Entry Attributes (QHFCHGAT) APIs.

#### **QACCDTTM**

CHAR(13)

The date and time of day the file or directory was last accessed, in CYYMMDDHHMMSS format.

You can specify this attribute on any API call, but your file system might ignore it in some APIs.

#### **QWRDTTM**

CHAR(13)

The date and time of day the file or directory was last written to, in CYYMMDDHHMMSS format.

Changes to this attribute may not be supported by all file systems. Refer to the file system's documentation for any restrictions.

#### **QFILATTR**

CHAR(10)

The type of item the directory entry is for. You can specify this attribute on any API call, except as noted in the following list of character descriptions.

Each character in the QFILATTR attribute has a specific meaning. Characters 6-10 must be set to blanks. Characters 1-5 must have a value of either 0 or 1:

0	No
1	Yes

The characters and their meanings are:

1	Read-only file. This applies only to files; it is ignored for directories. You can change it only by using the Change Directory Entry Attributes (QHFCHGAT) API.  When a file has this attribute, the file cannot be accessed in write mode. It cannot be opened with write or read/write access, it cannot be the target file in a copy stream file operation, and it cannot be deleted.
2	Hidden file or directory. You can change this attribute by using the Change Directory Entry Attributes (QHFCHGAT) API.
3	System file or directory. You can change this attribute by using the Change Directory Entry Attributes (QHFCHGAT) API.
4	Entry is a directory (not a file). You cannot change this attribute. If you specify it on the Change Directory Entry Attributes (QHFCHGAT) API, it is ignored.
5	Changed file. This applies only to files. It indicates that the file has been changed and is usually used to determine when a file needs to be moved to safe, permanent storage. It is set to Yes when the file is created or written to. You can set it to No only by using the Change Directory Entry Attributes (QHFCHGAT) API.
6-10	Reserved. Must be set to blanks.

## QERROR

CHAR(7)

A special attribute that can be returned in the attribute data buffer by the Read Directory (QHFRDDR) API when it encounters an error in retrieving the attributes of a directory entry. These values can be returned:

Message ID	Error Message Text
CPF1F06	Directory in use.
CPF1F06	Directory in use.
CPF1F08	Damaged directory.
CPF1F26	File in use.
CPF1F28	Damaged file.
CPF1F62	Requested function failed.
CPF1F71	File system unique exception occurred.

For details about calling the QHFRDDR API from an application, see the “Read Directory Entries (QHFRDDR) API” on page 46. For details about the interface between the QHFRDDR API and a new file system, see the “Exit Program for Read Directory Entries (QHFRDDR) API” on page 93.

## Other HFS Directory Entry Attributes

File systems can define their own unique attributes for files and directories in addition to the standard ones. The file system’s documentation defines the attributes’ names and values, and explains how to access and use them.

An application can also define its own directory entry attributes. These attributes are sometimes called **extended attributes**. They resemble the extended attributes in the IBM<sup>(R)</sup> OS/2<sup>(R)</sup> file system and supply additional information relevant to the application. The application must define the names and values of these attributes.

## HFS Attribute Information Table

The HFS APIs use a common attribute information table to pass all types of directory entry attributes between the application and the file system. Thus, the information is returned in the same format regardless of which file system the application is using. Different file systems can use different attributes, so the contents of the table can vary from one file system to another.

The file system must use the attribute information table when communicating with the application. It must accept and return attributes in that format. For its own use, however, the file system can store the attribute information in whatever format is most convenient.

The table consists of zero or more attributes and varies in length.

The attribute information table is used by these HFS APIs:

- Create Directory (QHFCRTDR)
- Retrieve Directory Entry Attributes (QHFRTVAT)
- Change Directory Entry Attributes (QHFCCHGAT)
- Read Directory Entries (QHFRDDR)
- Open Stream File (QHFOPNSF)

The attribute information table has three logical parts:

1. The first field specifies the number of attributes defined in the table.
2. The next fields give the offsets to the attributes defined in the table. There is one offset field for each attribute.
3. Next are groups of fields describing the attributes being defined or retrieved. There is one group of descriptive fields for each attribute.

The format of the attribute information table is:

Type	Field
BINARY(4)	The number of attributes defined in the table. This is the number of attributes being defined for or retrieved from the directory entry.
<i>Offsets:</i>	
BINARY(4)	The offset to the first attribute
BINARY(4)	The offset to the next attribute, if more than one is being defined or retrieved. This field is repeated to list the offset to each attribute being defined or retrieved.
<i>Description of the first attribute:</i>	
BINARY(4)	Length of attribute name
BINARY(4)	Length of attribute value
BINARY(4)	Reserved; currently set to zero
CHAR(*)	Attribute name
CHAR(*)	Attribute value <sup>1</sup>
<i>Description of the next attribute (repeated for each attribute after the first):</i>	
BINARY(4)	Length of attribute name
BINARY(4)	Length of attribute value
BINARY(4)	Reserved; currently set to zero
CHAR(*)	Attribute name



Type	Field
CHAR(*)	Attribute value <sup>1</sup>
<b>Note:</b>	
<sup>1</sup> The attribute value is either a character or the character representation of a binary field.	

## HFS Attribute Selection Table

The HFS APIs use a common attribute selection table to choose which directory entry attributes to retrieve or to make available for reading. The attribute selection table specifies which attributes the file system should return to the application. The file system must read the table, determine which attributes have been selected, and return those attributes to the application in the attribute information table.

The attribute selection table varies in length.

The attribute selection table is used by these HFS APIs:

- Open Directory (QHFOPNDR)
- Retrieve Directory Entry Attributes (QHFRTVAT)

The attribute selection table contains an entry for every attribute the application selects. If a selected attribute does not exist for the directory entry, no error is signaled. The attribute name is returned, and the length of the attribute value is zero.

The attribute selection table has three logical parts:

1. The first field specifies the number of attributes specified in the table.
2. The next fields give the offsets to the attributes specified in the table. There is one offset field for each attribute.
3. Next are pairs of fields describing the attributes specified. There is one pair of descriptive fields for each attribute.

The format of the attribute selection table is:

Type	Field
BINARY(4)	The number of attributes specified in this table
<i>Offsets:</i>	
BINARY(4)	The offset to the first attribute
BINARY(4)	The offset to the next attribute, if more than one is specified. This field is repeated for each attribute.
<i>Description of the first attribute:</i>	
BINARY(4)	Length of attribute name
CHAR(*)	Attribute name
<i>Description of the next attribute (repeated for each attribute specified):</i>	
BINARY(4)	Length of attribute name
CHAR(*)	Attribute name

---

## New File Systems

Two types of tools are necessary to use your own file system with the HFS APIs. You must supply these tools:

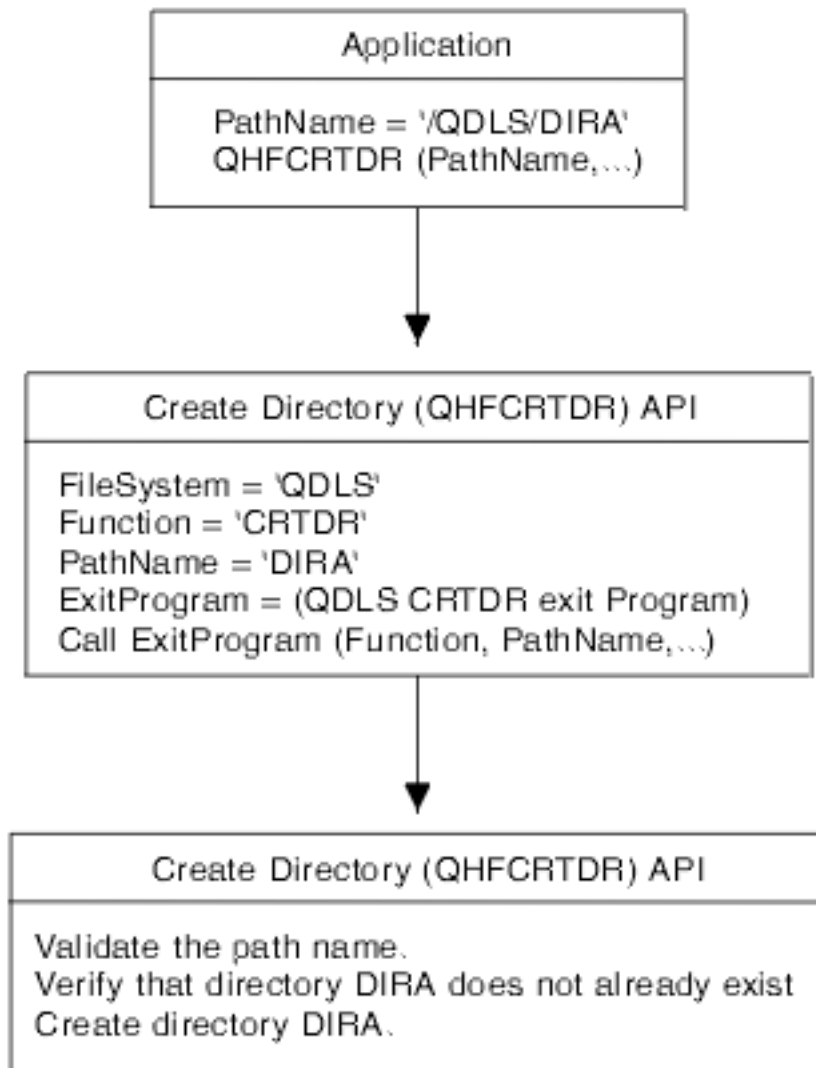
- Exit programs that do the work for the HFS APIs. You must provide these exit programs. **Exit programs** enable application programs to use the specific HFS APIs to work with files and directories in your file system. The HFS Exit Program—Scenario that follows shows how an application creates a directory in your file system using an exit program.
- Programs that call the Register File System (QHFRGFS) and Deregister File System (QHFDREGFS) APIs. Registration makes your file system and its exit programs accessible to application programs using the HFS APIs. Deregistration lets you remove a file system from use.

### Scenario: HFS Exit Program

Most exit programs correspond directly to APIs. For example, to allow applications to create directories in your file system, you must write an exit program that supplies the Create Directory function (CRTDR) for the Create Directory (QHFCRTDR) API. When an application calls an HFS API, the system routes the call to the appropriate exit program in the appropriate file system, according to the API's function and the file system name specified in the API's path-name parameter.

**Note:** The abbreviation CRTDR in the following illustration is the function name. See “File System Registration APIs” on page 51 for more information about function names.

## Creating a Directory in Your File System



[Top](#) | ["Hierarchical File System APIs," on page 1](#) | [APIs by category](#)

---

## Enabling Your File System to HFS

To make your file system available for use with the HFS APIs, take these steps:

1. Read "Standard HFS API and Exit Program Functions" on page 117 carefully. It describes which functions the HFS APIs perform for you and which functions your file system's exit programs must perform.
2. Define your own file and directory objects. These objects can be any of the following:
  - OS/400<sup>(R)</sup> objects, such as user spaces and user indexes
  - Objects on external devices attached to your iSeries<sup>(TM)</sup> server
  - Objects on other systems that are attached to your iSeries server by communications lines

Your file system controls the structure, format, and location of the file and directory objects it defines. It also controls security and authority for those objects.

### 3. Create these programs:

- A program or command to call the Register File System (QHFRGFS) API, which enrolls the file system for use with the HFS APIs.
- A Start Job Session exit program, which is called the first time a job tries to use the file system. See “Start Job Session Exit Program” on page 105 for details.
- An End Job Session exit program, which is called at job end to perform job cleanup. See “End Job Session Exit Program” on page 79 for details.
- An exit program for every HFS API function that you want your file system to support. For a list of exit programs you can supply, see the list of qualified exit program names parameter in the “Register File System (QHFRGFS) API” on page 54. If your file system is written in a high-level language that supports a variable number of input parameters, you can specify the same exit program for more than one function.

Your file system does not need to support every function. However, if you supply an Open Stream File or Open Directory function to support the QHFOPNSF or QHFOPNDR API, you must supply the corresponding close function.

4. Register your file system with HFS, specifying the exit programs you want to make available. See “Register File System (QHFRGFS) API” on page 54 for complete instructions.
5. Give the file system’s users authority to the “Start Job Session Exit Program” on page 105. Authority to this program gives users authority to the file system as a whole.
6. Give the file system’s users authority to the HFS APIs that you support with exit programs and that you want to make public. A user needs \*USE authority to an API to call the API from a program.
7. Provide the file system’s users with complete documentation, so they know which HFS APIs are supported, and whether any of the HFS APIs work differently from the way they are described in this information.

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## How HFS Support Processes a File System Job

OS/400<sup>(R)</sup> HFS support and your file system work together to perform the function that an application requests when it calls an HFS API. **HFS support** is the part of the system that manages the HFS APIs as a group and passes information between the HFS APIs and the file system.

### First Call to File System

The first time an application or job specifies a particular file system in a call to an HFS API, HFS support does the following:

1. Checks the job’s authority to the file system by checking its authority to the Start Job Session exit program. The job must have at least \*USE authority to the Start Job Session exit program.
2. Obtains a shared read (\*SHRRD) lock on the Start Job Session exit program. This prevents other jobs from deregistering the file system while the current job is using it. This lock remains in effect until the current job ends.
3. Calls the Start Job Session exit program, supplying the file system’s name on the call in case the file system has been renamed during installation.

The Start Job Session exit program performs any setup that the file system requires and returns a job handle to HFS support. The job handle is an arbitrary identifier that HFS support passes to the file system to help the file system keep track of this job. If the job calls another HFS API for this file system, HFS support passes this job handle as a parameter. The handle is treated as if it were a pointer, but it does not have to contain pointer data.

4. Calls the file system to complete the job’s request by performing the work for the API.

## Subsequent Calls to File System

On subsequent calls to the file system, HFS support retrieves the job handle for that file system and calls the file system exit program for the appropriate API. The file system can use the Start Job Session exit program only at the start of a job.

## End of Job

When OS/400 work management notifies HFS support that the job has ended, HFS support checks to see if the job has left any files or directories open. If it has, HFS support calls the file system to close them. HFS support then calls the End Job Session exit program to clean up any work areas used by the job. Your file system can use the End Job Session exit program to destroy temporary spaces and remove outstanding locks that were created during the job.

After the End Job Session exit program is run, HFS support unlocks the Start Job Session exit program, which was locked when the job first called the file system through an HFS API. The file system is no longer in use and can be deregistered.

## Scenario: File System Job

When the Start Job Session exit program is called, the file system could create a user space in the job's QTEMP library and return a space pointer to that user space as the job handle. On subsequent calls to HFS APIs for that file system, HFS support would pass that space pointer to the file system as the job handle. When the job ends, the file system's End Job Session exit program is called. The file system could use that exit program to delete the user space.

## File System—Error Messages

The exceptions that file systems are allowed to use are listed after each of the HFS exit programs. When a file system returns an allowed exception after a call from an HFS API, HFS support either sends the exception to the application or fills in the error code.

[Top](#) | [“Hierarchical File System APIs,” on page 1](#) | [APIs by category](#)

---

## Standard HFS API and Exit Program Functions

The HFS APIs and the file system's exit programs share the responsibility for validating input data, performing the functions requested by the application, reporting errors, and so on. The two sections that follow describe the standard functions that the HFS APIs perform for you and the standard functions that your exit programs must perform. A few APIs perform additional functions, and a few exit programs have additional requirements; these additions are listed at the end of each exit program description.

## Standard HFS API Functions

This section describes the functions that the HFS APIs and HFS support perform for your file system. When an application calls an HFS API, it appears to the application that the HFS API performs all the resulting functions. In reality, the API and HFS support perform only some of the functions. The file system exit program performs the rest.

Use this information to plan and create exit programs to support the HFS APIs in new file systems. You do not need this information to use the HFS APIs in your applications.

In general, every HFS API performs these functions for your file system:

- Automatically calls the file system's Start Job Session exit program the first time a job refers to the file system in a call to an HFS API. For a detailed explanation, see “Start Job Session Exit Program” on page 105.
- Processes the path name parameter as follows:

- Extracts the file system name from the path name parameter.
- Verifies that the file system is registered for use with the HFS APIs.
- Passes the part of the path name that follows the file system name to the file system. For example, if the path name received from the application is /QDLS/A/B/C, then /A/B/C is passed to file system QDLS.  
There is one exception to this: The Open Directory (QHFOPNDR) API allows jobs to open the directory representing the file system itself. In that case, the path name consists of a slash and a single element, such as /QDLS, and the API passes just the slash (/) to the file system.
- Recomputes the length of the path name by subtracting the length of the extracted file system name, and passes the new length to the file system.
- Verifies that there is at least one valid byte (the leading /) in the rest of the path name parameter.
- Verifies that a directory or file handle passed from the application to the API is valid, and looks up the corresponding directory or file handle to pass from the API to the file system.
- Verifies that API parameters contain allowable values, and that reserved portions are set to blanks. API parameters include all fixed-length character fields like the open information parameter of the Open Stream File (QHFOPNFS) API.
- Verifies that length parameters contain allowable values. These length parameters are verified:
  - Length of attribute selection table
  - Length of attribute information table
  - Length of data buffer to read directory entries into
- Verifies that numeric or counting parameters contain a valid value. These parameters are verified:
  - Number of directory entries to read
  - Number of bytes to read
  - Number of bytes to write
- Calls the appropriate file system exit program to perform the operation.
- Monitors for valid exceptions from the file system, and either returns these to the application or maps them into an error code, as the application requests in the API's error code parameter.
- Signals successful completion of the operation to the application by not returning any errors.
- Returns control to the application.

## Standard HFS Exit Program Requirements

In general, every exit program you provide to support an HFS API must perform these functions:

- Verifies that the application has authority to perform the requested operation on the specified objects.
- Verifies that parameter values meet the file system's criteria.
- Verifies that directories and files named in the path name parameter either exist in the file system or, during operations like create and rename, that new names conform to the file system's naming conventions.

The path name that the API passes to the file system does not include the file system's name but only the rest of the path name. If an application calls the Open Directory (QHFOPNDR) API and passes a path name specifying only the directory consisting of the file system, such as /QDLS, the API passes only the slash (/) to the file system.

- Maintains API INPUT parameters as is, without changing their contents. If the file system must change these parameters, it must move them to another storage location. INPUT parameters must have the same value on entry to the file system exit program as on exit from the exit program. The file system can change OUTPUT parameters when the requested operation succeeds.
- Accepts attributes in the attribute selection and information tables used by HFS support, validates the data contained in the tables, and returns the appropriate data in the appropriate format.

HFS support uses two common formats for passing attribute information between the application and the file system. These formats are described in “HFS Attribute Selection Table” on page 113 and “HFS Attribute Information Table” on page 112.

- Performs the requested operation and returns any requested status information or data to the API.
- If the operation does not succeed, returns an exception describing the error to the API.

The file system should return only the exceptions defined for the API because HFS support monitors only for those messages. If the file system sends any other message, the message Internal file system error occurred is returned to the application.

If the file system needs to send its own messages, it can use the message Exception specific to file system occurred defined for each API. The file system’s message ID is sent as insert data.

[Top](#) | [“Hierarchical File System APIs,”](#) on page 1 | [APIs by category](#)





---

## 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 grant 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 information 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.

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.

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

---

## Trademarks

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

Advanced 36  
Advanced Function Printing  
Advanced Peer-to-Peer Networking  
AFP  
AIX  
AS/400  
COBOL/400  
CUA  
DB2  
DB2 Universal Database  
Distributed Relational Database Architecture  
Domino  
DPI

DRDA  
eServer  
GDDM  
IBM  
Integrated Language Environment  
Intelligent Printer Data Stream  
IPDS  
iSeries  
Lotus Notes  
MVS  
Netfinity  
Net.Data  
NetView  
Notes  
OfficeVision  
Operating System/2  
Operating System/400  
OS/2  
OS/400  
PartnerWorld  
PowerPC  
PrintManager  
Print Services Facility  
RISC System/6000  
RPG/400  
RS/6000  
SAA  
SecureWay  
System/36  
System/370  
System/38  
System/390  
VisualAge  
WebSphere  
xSeries

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

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. 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 for downloading and printing publications

Permissions for the use of the information you have selected for download are granted subject to the following terms and conditions and your indication of acceptance thereof.

**Personal Use:** You may reproduce this information for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of this information, or any portion thereof, without the express consent of IBM<sup>(R)</sup>.

**Commercial Use:** You may reproduce, distribute and display this information solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of this information, or reproduce, distribute or display this information 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 information or any 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 information 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 THIS INFORMATION. THE INFORMATION IS 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.

All material copyrighted by IBM Corporation.

By downloading or printing information from this site, you have indicated your agreement with these terms and conditions.

---

## **Code disclaimer information**

This document contains programming examples.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM<sup>(R)</sup>, 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. 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 INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.





Printed in USA