HFS FOR z/VSE

User Guide and Installation Notes

Hierarchical File System

Release 2.3

HFS allows users to store files on the IBM z/VSE mainframe in a hierarchical manner similar to that used on PCs and LINUX-based systems. HFS's File Interception Facility also performs file and format conversions.

CSI INTERNATIONAL

"Delivering what the competition can only promise"

www.csi-international.com • info@csi-international.com • 800.795.4914

HFS FOR z/VSE User Guide and Installation Notes
Release 2.3
First edition July 2006

Published by CSI International

Phone: 800-795-4914

Internet: http://www.csi-international.com

Product questions: info@csi-international.com

Technical support: support@csi-international.com

Table of Contents

Introduction

HFS allows users of this product to store files on the IBM z/VSE mainframe in a hierarchical manner like that used on PCs and LINUX-based systems. HFS allows users to:

- Organize files in a convenient, hierarchical manner
- Use more meaningful, long file names that include embedded spaces
- Use as many levels of directories and subdirectories as necessary to organize data
- Use a CICS transaction to view and maintain HFS files

In addition, if you want to take advantage of the File Interception Facility of HFS, you can:

- Encrypt your data using DES, DES3 or AES128 algorithms
- Store and retrieve files in a hierarchical manner
- Process fixed-length data records
- Process variable-length data records
- Generate AWS format output
- Convert from mainframe format to comma separated strings
- Convert from comma separated strings to mainframe format
- Convert from mainframe format to LINUX-like binary images
- Convert from LINUX-like binary images to mainframe format
- Generate HTML or XHTML output
- Generate XML output
- Process comma-separated, CRLF-delimited strings directly.

And it does all of this without requiring a single change to your existing legacy programs—you don't even have to recompile them; all you need to do is make a simple change to the JCL, and HFS will take over.

The HFS File Interception Facility can:

- Intercept I/O to Sequential Disk Files (DTFSD)
- Intercept assembler DTFs
- Intercept files for both FCOBOL (old) and LE/COBOL (current)

Provided that z/VSE DTFs are used in a standard manner, the HFS Interception Facility should work with any program in your shop, including those provided by software vendors other than CSI International.

Overview

LINUX has it, the PC on your desktop has it, now z/VSE has it too. With HFS from CSI International, you too can store your data in a hierarchical structure on the mainframe. And you can use long file names (256 bytes) as well.

Let us take a look at a sample data structure:

```
<DIR> PRODUCTION
      <DIR> Revenue Accounting
            <DIR> Accounts Payable
                  Week to Date Payables (RA4104)      6,456,756
                  Month to Date Payables (RA4106)    22,325,102
                  . . .
            <DIR> Accounts Receivable
                  Week to Date Receivables (RA4112) 104,722,345
                  . . .
<DIR> TEST
      <DIR> Revenue Accounting
            <DIR> Accounts Payable
                  Week to Date Payables (RA4104)        234,716
                  Month to Date Payables (RA4106)       865,112
                  . . .
```

To get to "Month to Date Payables," the fully qualified path name is "/PRODUCTION/Revenue Accounting/Month to Date Payables (RA4106)" just like it is on a PC. To switch to test data, change the high-order directory: "**/TEST**/Revenue Accounting/Month to Date Payables (RA4106)."

Perhaps you need to organize your data by division or location. You can easily do that, too:

```
<DIR> Chicago
      <DIR> Revenue Accounting
            <DIR> Accounts Payable
                  Week to Date Payables           13,234
                  . . .
            <DIR> Accounts Receivable
      . . .
<DIR> Minneapolis
      <DIR> Revenue Accounting
            <DIR> Accounts Payable
                  Week to Date Payables           87,313
                  . . .
            <DIR> Accounts Receivable
      . . .
```

As you will see later, with the HFS File Interception Facility it is easy to switch between hierarchies in an HFS.

What kind of data can be stored in an HFS? Anything you want. At CSI, we have stored and retrieved JPEGs., GIFs, HTML, JAVA, ASCII text, EBCDIC text, core dumps, reports, binary files—literally any kind of data file you have. CSI's website, www.csi-international.com, runs on a z/VSE mainframe with *Entrée* and uses HFS to store its pages, scripts, and downloads.

With *TCP/IP for VSE* release 1.5E and higher, you can transfer data from an HFS to anywhere in your network, and likewise you can move data into an HFS from anywhere.

But, the ability to organize data into a hierarchical structure isn't much good unless you can readily access it with your existing legacy applications, which brings us to …

HFS provides transparent access to its features from your legacy application programs.

You don't have to change a line of code, you don't have to add a line of code, you don't even have to recompile your programs in order to access data stored in an HFS – a simple JCL change will suffice. For example:

```
* $$ JOB JNM=somejob, . . .
// JOB somejob
// DLBL . . .     whatever you need
// EXTENT . . .
// DLBL HFSfile,'file name',0,DA
// EXTENT  . . .
// EXEC CSIHLABL
 HLBL OUTFILE,'/Chicago/Revenue Accounting …',0,SD, -
      HFS=HFSfile
/*
// EXEC YOURPROGRAM
 . . .
/&
* $$ EOJ
```

Now, when your program opens file "OUTFILE", HFS latches on to it and processes accordingly. If "OUTFILE" is opened for output, data is written to the HFS. If "OUTFILE" is opened for input, data is read from the HFS and returns it to your program one record at a time, just like VSE LIOCS.

"OUTFILE" in the above example can be a sequential disk file (DTFSD). HFS has been tested with assembler programs, COBOL programs compiled with FCOBOL (old) or LE/COBOL (current).

Provided that they use standard DTFSD processing, programs supplied by other software vendors should work seamlessly with HFS files.

Data Encryption is easily accomplished. Add the "SECURE" operand to the HLBL and the file will be encrypted in the HFS using either the DES, TDES or AES128 encryption algorithm – your choice. Naturally, the file will be decrypted when it is read back in.

File Conversion. Using an easy to understand script process, HFS can convert data from and to comma separated strings. LINUXX-like binary files can easily be created. Data can be converted to HTML, XHTML or XML if need be. HFS can even generate AWS tapes for transfer to other systems or for data archival.

And remember, HFS provides these features without requiring any program changes on your part.

Release Changes

Release 2.2

- Add DISP(old|new …) to HLBL
- Provide support for *HFS-Lite*

Release 2.1

- LIST command in CSIHFBAT is now sorted and supports several new formats
- DIR command in CSIHFBAT is now sorted and supports several new formats
- Added FIND command to CSIHFBAT
- Added TREE command to CSIHFBAT
- Added APPEND option to HLBL
- Added RENAME option to HLBL
- Added %DATE and %TIME variables to HLBL name generation
- Provided for generic selection for input files using the HFS File Intercept Feature.
- Original date and time are preserved for file backup and restore
- Bug fixes and corrections thus uncovering new bugs

Release 2.0

- Added HFS File Intercept Feature (HLBL)
- Added Load of SD files into HFS
- Added Load of LIBR files into HFS

HFS-Lite

HFS-Lite consists of a restricted subset of the entire **HFS** system.

HFS-Lite provides sufficient capability for you to explore the basic file structure of **HFS** but does not include all of features described in this document. Those parts of **HFS** that are not included with ***HFS-Lite*** are indicated in the document below.

You will need to separately purchase **HFS** in order to take advantage of the complete and rich feature set of **HFS**.

The following table summarizes the differences between **HFS** and ***HFS-Lite***.

Feature	**HFS**	***HFS-Lite***
Create HFS Extents	Yes	Yes – single extents are restricted to 14 3390 cyl equivalents.
Create multiple HFS extents	Yes	Yes
Manipulate data in HFS	Yes	Yes
Journal for forward recovery	Yes	No
Cache HFS for improved performance	Yes	No
Recover damaged HFS data	Yes	Yes
Use the HLBL process to write data using a standard VSE DTFSD	Yes	No
File Conversion	Yes	No
Application Programming Interface (API)	Yes	No

HFS File Structure

HFS Extent Allocation

An HFS extent (the HFS file) consists of 4096-byte records accessed by relative record number using physical IOCS. Each EXTENT must contain at least one full cylinder, or cylinder equivalent of DASD space.

An individual HFS extent can manage a maximum of 32,504,864 (x'01EFFC20') records (992 FAT records * 32,768 records/FAT record). Each record is 4096 bytes, so a maximum file can be about 133 gigabytes in size, which should be large enough for most applications. If this is not large enough, the HFS can manage several files for the same partition (currently a maximum of 64 files, for a total capacity of just under 4 terabytes).

Before an HFS extent can be used and files added to the HFS you must initialize the file with the INITIALIZE command as described in "HFS File Management," below.

HFS and Multiple Partitions

Multiple partitions can be used with HFS, and the system will coordinate add and update activity between the partitions to prevent collisions. However, each partition *must* use the same DLBL name for the HFS EXTENT. This restriction is enforced, and error message 312 will be issued if an attempt is made to open the same file a second time with a different DLBL name.

Note: HFS extents cannot be shared between CPUs.

File Name Length

File and directory names can be up to 256 bytes long. This is the fully qualified path name and includes the file name and any subdirectories comprising the name, followed by an optional extension up to 8 bytes long.

Periods and Name Extensions

Periods are legal anywhere in file names as they are on PCs. The system checks for an extension by scanning the name from right to left. If a period is found after 8 or fewer bytes have been scanned, the string to the right of the period is treated as an extension. Users should be careful when including periods in names so that unintended extensions are not created.

HFS makes no assumptions about file contents based on a file name's extension. Extensions only have meaning in relation to other products and as understood by users.

Restricted Characters

The following characters are not permitted in HFS file names:

* (asterisk), ? (question mark), \ (back slash), | (vertical bar),
/ (forward slash), : (colon), " (double quote), > (greater than), or
< (less than).

All other characters from the EBCDIC character set are allowed, including embedded spaces. These are the same restrictions that apply to Internet URLs.

Note: Except for asterisk and question mark, these restrictions can be overridden by using the OPTION batch command with the ALLOWNAME argument. This command is described later in this manual.

Path Names

File names passed to HFS can be in the form of a path name. That is, HFS will examine the file name and work its way through any embedded directories it finds. For example, if you are positioned at "/directoryOne", you can ask to read file "/directoryTwo/subdirOne/file-name" without having to establish the current working directory. Generally, the current working directory remains unchanged.

It is not necessary to specify the full path name in order to access or create a file. For example, suppose again you are positioned at "/directoryOne". The names "/directoryOne/file-name" and "file-name" will both yield the same file.

HFS also maintains a special directory entry named "..". This internal directory entry is intended for recovery purposes and will not appear in response to a directory request. It may, however, be useful when navigating through a file's directory tree. It has the same purpose as the ".." directory on a PC. For example, using a name of "../file-name" will locate the file in the immediately higher level directory.

File Locking

Whenever a file is accessed, it is locked to prevent damage from some other partition (or task, in the case of a multi-tasking system such as CICS or TCP/IP). There are two types of locks: read and write. Their interactions are as follows:

Read	When a file is locked for read, other partitions (tasks) can also read it. Write, Rename and Delete activity are prohibited.
Write	When a file is locked for write, no other partition (task) can access it for any purpose.

File locks are tracked globally in the VSE machine. That is, each task will check all locks, not just its own, before permission to access the file is granted. The file is locked when it is first opened for read or write processing. The lock is removed when the file is closed.

The Write lock does not apply to a newly created file because, by design, it is not accessible until closed. The last thing written to a new file is its directory entry, so it is simply not possible to access a file until it has been properly closed by the creating task.

Lock Removal

It is possible that locks may remain active when an HFS-related partition is abnormally terminated. One way to correct the locks that may remain from an abnormal termination is to add a job containing the RELEASE command to a batch file running other commands. The RELEASE command and a batch file example is described later in this manual.

HFS Operations

If you are using HFS as a companion product for other CSI offerings, such as *Entrée*, HFS requirements should be described in the manual that came with the other CSI product. Even if you do not intend to use HFS by itself, you should review the information presented below.

Requirements

1. To use FTP with HFS files, you must be running *TCP/IP for VSE* release 1.5E or higher

2. You must create a control file with a minimum of one cylinder of 3390 (or 3390 equivalent) DASD space. If you intend to make use of the file conversion facility of HFS you will probably need more DASD space than this.

   This file is named "HFSGEN" and to simplify operations its DLBL and EXTENT should be placed in System Standard Labels. The HFSGEN file must be initialized prior to use. Use the JCL below as a guide for initializing the file.

```
// DLBL HFSGEN, …      if needed
// EXTENT …            if needed
// LIBDEF *,SEARCH=(     as needed
// EXEC CSIHFBAT
 INITIALIZE HFSGEN /*INITIALIZE MUST STAND ALONE*/
/*
// EXEC CSIHFBAT
 ACCESS HFSGEN   /* CREATE DIRECTORIES FOR */
 MD '/$$FILE'    /* FILE DEFINITION */
 MD '/$$CONV'    /* CONVERSION SCRIPTS */
 MD '/$$XLAT'    /* TRANSLATION TABLES */
/*
```

   The three "MD" (Make Directory) commands must be supplied as shown above, the comments, of course, are optional.

   Management of HFS files is described in more detail below.

3. IF you will be using the File Interception Facility of HFS, you will need to supply a dummy DLBL and EXTENT for HFS' use. The EXTENT need only encompass one track. Nothing will ever be written to it or read from it but it is needed to satisfy COBOL's pre-open label checking routines. To simplify operations its DLBL and EXTENT should be placed in System Standard Labels.

```
// DLBL HFS$$$,'DUMMY FILE',0,SD
// EXTENT SYSnnn …
```

4. IF you will be using the File Interception Facility of HFS, you must ensure that phase CSIHLBLO is loaded into the SVA. This step should be added to your VSE IPL procedures. Use the JCL below as a guide to loading this phase into the SVA.

```
// LIBDEF *,SEARCH=(     as needed
SET SDL
CSIHLBLO,SVA
```

5. If you will be using the HFS File Interception Facility, you must have HFS Journaling active in a VSE partition. Even if you do not use the File Interception Facility, HFS journaling can provide siginificant performance improvements and its use is encouraged. This can be run in either a static or dynamic partition. (See "HFS Journaling" below.)

6. You will need to define one or more HFS extents to contain the files processed by HFS (See "Data Files" below.)

7. For each jobstream you wish to use with HFS File Interception Facility you will need to define one or more HLBLs. HLBLs are very similar to DLBLs and are described in more detail below.

Operations

For the HFS File Interception Facility to work properly, HFS Journaling must be active. Activating Journaling will use JCL that looks like:

```
* $$ JOB JNM=HFSJRNL,CLASS=G,DISP=K
* $$ LST CLASS=L
// JOB HFSJRNL
// LIBDEF *,SEARCH=(         as needed
// EXEC CSIHFJRN
 HFS CACHE  ( HFSGEN  SIZE( 100 )  )
 HFS HLABL  ( 500 ) /* label space for File Interception */
/*
/&
* $$ EOJ
```

HFS journaling is described in detail below.

Note, however, in the example above we also activated HFS Caching for the HFSGEN file. You will probably want to activate caching for the data file(s) you define as well.

Each individual data file you create must be first initialized using CSIHFBAT (discussed below under "HFS File Management."

The amount of DASD allocated and the number of individual HFS extents is up to you. Each HFS must be a minimum of one cylinder or cylinder equivalent of DASD. Other restrictions were discussed above.

For performance purposes you will probably want to cache the data file(s). See “HFS File Journaling” below for more information.

The HFS File Interception Facility will not generate sub-directories for you. You will need to establish the directory structure yourself using the batch commands available with HFS. For example, the following sample JCL initializes a data file and creates a directory structure for subsequent use.

```
// DLBL HFS01,'HFS01 DATA FILE',99/365,DA
// EXTENT SYSnnn …
// LIBDEF *,SEARCH=( …
// EXEC CSIHFBAT
 INITIALIZE HFS01
/*
// EXEC CSIHFBAT
 ACCESS HFS01
 MD '/ACCOUNTING FILES'     /* MAKE ACCOUTING FILES */
 MD '/ACCOUNTING FILES/DAILY'
 MD '/ACCOUNTING FILES/WEEKLY'
 MD '/ACCOUNTING FILES/MONTHLY'
 MD '/ACCOUNTING FILES/MONTHLY/MONTH TO DATE'
 MD '/OTHER FILES'          /* MAKE OTHER FILES */
 MD '/OTHER FILES/MISC'
    . . .
/*
```

The sample JCL fragment below illustrates what is needed to encrypt a data file using HFS File Interception Facility.

```
   . . .
   Your DLBLs and TLBLs here
// DLBL HFS$$$$,'DUMMY FILE',0,SD     should be in
// EXTENT SYSnnn, …                   STDLABELS
// DLBL HFS01,'…                         "
// EXTENT SYSnnn, …                      "
// LIBDEF *,SEARCH=(                  as needed
// EXEC CSIHLABL,SIZE=AUTO
 HLBL CSITEST '/FIXTST FILE' 0 SD HFS=HFS01  -
             SECURE( DFLT )
/*
// EXEC your.program
   . . .
```

In this example, your program is assumed to be creating a file named "CSITEST." By adding the CSIHLABL step you inform HFS that it is to take over I/O for this file and direct it instead to HFS01 under the name "/FIXTST FILE" and to encrypt it using the default encryption method and key.

Assuming the DLBL and EXTENTs are included in Standard Labels, the bolded lines are the only ones needed to produce encrypted output.

Periodic Maintenance

Some garbage may collect in an HFS over time. The most common cause for this is when using the HFS File Interception Facility and your program for some reason, abend or otherwise, does not properly close an output file. In this scenario, one or more garbage records may accumulate in the HFS.

This can be easily fixed using the HFS File Recovery process (described in detail below). You will need to occasionally run a job that looks something like:

```
* $$ JOB JNM=HFSRECOV,CLASS=W
// JOB HFSRECOV
// DLBL CSIHFDT,'HIERARCHICAL FILE',0,DA     99/365,DA
// EXTENT SYS009, . . .
// EXEC CSIHFBAT,SIZE=AUTO
 ACCESS CSIHFDT
 RECOVER DEEP NOAUTO NOTRACE
/*
/&
* $$ EOJ
```

You will probably need to run this often in a testing environment, but only occasionally in production.

HFS File Management

HFS File Management is done using a batch program, CSIHFBAT. This program is used for HFS initialization, backup, restore, and recovery. Also, batch commands can be used to manage the files and directories within the HFS.

The commands described in this section are listed under two categories, HFS Maintenance Commands and File Management Commands. The RECOVERY and JRNL commands and processes are described in a separate sections of this manual. Command syntax and documentation conventions are described separately in "Appendix A – Parameter Syntax."

Use the following JCL as a guide for running the CSIHFBAT program:

```
* $$ JOB JNM=HFSCMDS,CLASS=2
* $$ LST CLASS=L
// JOB HFSCMDS
// DLBL CSIHFDT,'HIERARCHICAL FILE',0,DA
// EXTENT SYS009, . . .       as needed
// ASSGN SYS009, . . .        as needed
// LIBDEF *,SEARCH= . . .     as needed
// EXEC CSIHFBAT,SIZE=AUTO
ACCESS CSIHFDT
DIR
LIST
STATS
/*
/&
* $$ EOJ
```

Other than for the INITIALIZE command (see below), the ACCESS command must be present and must be the first command encountered by CSIHFBAT.

HFS Maintenance Commands

ACCESS

ACCESS hfs.file [JOURNAL | NOJOURNAL] [SEP(char)]

Specifies the HFS file to process. This command must precede all other commands (except INITIALIZE) in batch file programs.

hfs.file	Specifies the DLBL name of the HFS file to be used.
JOURNAL|NOJOURNAL	Indicates whether changes from this execution of CSIHFBAT are to be journaled.
SEP(char)	Can be used to change the directory separator character from the default, forward slash (‘/’), to any other character except for period (‘.’), which is reserved for internal use in HFS for the file name extension.

This command can be abbreviated as **ACC**.

BACKUP

BACKUP {DISK(filename) | TAPE(filename) } - [DETAIL | NODETAIL] – [UNLOAD | NOUNLOAD] – [PHYSICAL | NOPHYSICAL]

Backs up the HFS (the HFS file) to *filename*.

DISK(filename)	Specifies that the backup is to be written to a disk file. Either TAPE or DISK must be specified.
TAPE(filename)	Specifies that the backup is to be written to a tape file. The tape file will be written on SYS004.
DETAIL | NODETAIL	Indicates the depth of information printed during the backup.
UNLOAD | NOUNLOAD	This operand applies to TAPE backup only and determines whether the tape will be unloaded on close or left positioned as is. This option can be used to stack files on a single tape volume.
PHYSICAL | NOPHYSICAL	Determines the type of backup to be created. A PHYSICAL backup copies physical records from the HFS file as is with no attempt to organize them into individual files. A NOPHYSICAL backup copies the HFS file by file (another term for this is logical backup). The type of backup you specify determines the type of restore that can be run.

You must use the PHYSICAL option for backups and restores intended for use with the JOURNAL RECOVER process. Attempting a JOURNAL RECOVER using the results from a restore of a NOPHYSICAL backup will result in unpredictable behavior and certain data loss.

Do not perform a NOPHYSICAL (logical) backup while HFS is active in other partitions or tasks. Doing so may produce an incomplete backup. Such a backup will not contain:

- **any portion of a file on HFS that is being written to while the backup is running. The file will be invisible to the BACKUP process.**
- **any file that is being updated at the time the BACKUP process is running.**

In addition, results are unpredictable for files being deleted or renamed during the BACKUP process.

INITIALIZE

INITIALIZE hfs.file [QUICK | NOQUICK]

Formats the HFS extent and prepares it for use with subsequent programs.

The INITIALIZE command must be run by itself in a single execution of CSIHFBAT.

hfs.file	This is the seven character DLBL name of the HFS file.
QUICK | NOQUICK	QUICK assumes that the file has been previously initialized with NOQUICK (the default). QUICK just resets the HFS file and establishes an empty ROOT directory. **If QUICK is used on an unformatted extent (not previously initialized), the HFS will fail in subsequent processing.**

FILELIST

FILELIST {DISK(filename) | TAPE(filename)}

Lists the contents of a NOPHYSICAL backup file that was generated by the BACKUP command.

DISK(filename)	Lists the contents of the named disk file.
TAPE(filename)	Lists the contents of the named tape file. The tape will be processed using SYS004.

Note: The FILELIST command will not work against a PHYSICAL backup.

RELEASE

RELEASE {partition}

Removes file locks for a partition.

partition	This is an optional parameter. If omitted, locks for the current partition will be removed. If specified, locks for the specified partition will be removed.

Individual HFS file locks will commonly be left hanging following an abend, or in a testing environment when a program does not properly close the file. One way to correct the locks that may remain is to add a separate job containing the RELEASE command to a batch file that runs other commands. This approach is shown in the following example:

```
// JOB xxxxx
// <whatever your job needs>
   . . .
/& Use a separate VSE job to ensure it gets executed on abend
// JOB FIXLOCKS
// DLBL hfsfile
// EXTENT
// EXEC CSIHFBAT
ACCESS hfsfile
RELEASE
/*
/&
* $$ EOJ
```

Repeat the ACCESS and RELEASE commands as often as needed to clean up files if more than one HFS extent is used in your process.

RESTORE

RESTORE {DISK(filename) | TAPE(filename)} -
[<u>REPLACE</u> | NOREPLACE] -
[<u>UNLOAD</u> | NOUNLOAD] -
[PHYSICAL | <u>NOPHYSICAL</u>]

Restores HFS files using data from a previous BACKUP command.

DISK(filename)	Specifies that the restore operation is to use the named disk file.
TAPE(filename)	Specifies that the restore operation is to use the named tape file.
<u>REPLACE</u> | NOREPLACE	Determines the processing used when a duplicate file name is encountered. Specify NOREPLACE if you do not want existing files replaced with information from a previous backup. This option is ignored for a PHYSICAL Restore. You can use this option to recover from an accidental deletion of one or more files. Simply run RESTORE using the NOREPLACE option, and any files found on the backup that are not currently on the HFS file will be replaced. All others will be bypassed.
<u>UNLOAD</u> | NOUNLOAD	Determines whether the tape will be unloaded on close or left positioned as is. NOUNLOAD leaves it positioned where it is so that if a subsequent file is present on the tape volume, the tape is positioned correctly to read it. This option is not relevant to a DISK backup.
PHYSICAL | <u>NOPHYSICAL</u>	Specifies the type of restore to run based on the type of backup being used. A physical restore copies physical records from the backup file as is with no attempt to organize them into individual files.

RESTORE must be the first command to follow the ACCESS command in a job. Other commands may follow the RESTORE command if needed. Failure to follow this restriction will result in unpredictable behavior.

You must run INITIALIZE on the HFS file before performing a NOPHYSICAL (logical) restore of an HFS. Otherwise, the HFS will fail on subsequent processing.

STATE

STATE

Provides dumps of internal storage blocks which can be useful in debugging. Use this command when requested by CSI Technical Support. There are no options for the STATE command.

STATS

STATS

Prints HFS file statistics. There are no options for the STATS command. The resulting output looks like:

```
SYSIN STATS
HBAT: FILE: C S I H F D T
HBAT: DATE: 0 3 / 1 4 / 0 5
HBAT: TIME: 0 7 : 0 2 : 4 4
HBAT: RECORDS MAX.:      7,800
HBAT: RECORDS USED:         25     0 0 . 3 2 %
HBAT: CACHE HITS:       12,456
HBAT: CACHE MISSES:        103
HBAT: FILE READS:           16
HBAT: FILE WRITES:           4
```

Explanation:

RECORDS MAX.	Maximum records in the HFS extent.
RECORDS USED:	Current number of records in use and the percentage used.
CACHE HITS:	Directory Records retrieved from caching (see “HFS File Journaling” below).
CACHE MISSES:	Directory reads made when a record could not be located in the cache.
FILE READS:	Total reads issued for this HFS extent.
FILE WRITES:	Total writes issued for this HFS extent.

UNLOCK

UNLOCK [ALL | NOALL] [SHOW | NOSHOW] - [FILE(name)] …

Individual files can be locked for either read or write access. This command can display these locks, and optionally, clear them up.

ALL | NOALL	UNLOCK ALL functions similar to the RELEASE command. Locks for the current partition will be removed.
SHOW | NOSHOW	Specify UNLOCK SHOW to receive a report of all locks for the current partition.
FILE(name)	Specifies a file to unlock. The FILE option may be repeated to select multiple files. The *name* must be the fully qualified path name for the file beginning with the ROOT directory.

WAKEUP

WAKEUP

Certain parts of the HFS are necessarily single-threaded. If problems occur, it is possible that the HFS can be left in a state where one or more partitions or subtasks are in an unending wait state. WAKEUP will free all outstanding waits that currently exist for the HFS.

Use this command with caution **and then only when directed by CSI Technical Support. WAKEUP will issue diagnostic information that should be forwarded to CSI Technical Support for analysis.**

File Management Commands

Use the following commands to manage files and directories within the HFS.

CHANGEDIR

CHANGEDIR name

Changes to the directory *name* as supplied with the command

This command can be abbreviated as **CD**.

COPY

COPY name1 TO(name2) [REPLACE | NOREPLACE].

Copies a file. Files can be copied within the same HFS extent, or they can be copied to a different HFS extent depending on the format of the name supplied

name1	Specifies the name of the HFS file that will be copied. This name is required.
name2	Specifies the target name for the copy operation. This name is required.
REPLACE | NOREPLACE	Governs processing when a duplicate file name is encountered for the TO name. REPLACE allows overwrites. NOREPLACE will not overwrite the file.

Names used in the COPY command can be made generic by using the following pattern-matching characters:

- ***** Matches any number of any characters
- **+** Matches any single character.

So, for instance, COPY '*.TXT' TO('/TEXT DIR/*.TXT') will copy all HFS files with an extension of ".TXT" from the current directory to "/TEXT DIR."

Optionally, the name of the HFS extent can be added to either *name1* or *name2* to access a different HFS extent than currently specified. For example,
`COPY 'FILE ONE' TO('HFS01:/DIRECTORYX/FILE ONE')` will place a copy of the input file into the "HFS01" extent.

DELETE

DELETE name

Deletes the file *name* from HFS.

The DELETE command cannot be used to delete a directory. For directories, use the REMOVEDIR command.

This command can be abbreviated as **DEL**.

DIRECTORY

DIRECTORY [STATS | NOSTATS] [SORT | NOSORT] – [FORMAT(3)] [MATCH('value')]

Prints a list of the contents of the named or current working directory.

STATS | NOSTATS	Specify STATS to obtain directory statistics. In the first example below, the last four lines are a result of the STATS option.
SORT | NOSORT	The HFS directory is not maintained in a sorted fashion. Specify NOSORT to print the directory in its actual order. Specify SORT (the default) to cause the directory to be sorted priot to printing.
FORMAT(3)	Optionally, you can retrieve this information in a second format – see examples below. The default is FORMAT(3).
MATCH('value')	This optional parameter can be used to select only a portion of the contents of the directory for display. Wild card values can be either: ***** - any number of any characters, or **+** - any single character. For example, "*.HT*" will select every file whose extension begins with "HT" – "HTM", "HTML", etc.

This command can be abbreviated as **DIR**.

Example of FORMAT(1) directory including STATS:

```
SYSIN DIR STATS
HFS: DIRECTORY OF : ROOT
HFS: VARIABLE TABLE FILE       2763
HFS: AWSTST FILE              26394
HFS: <DIR> RENAMED DIR
HFS: VARIABLE CONVERT FILE     2586
HFS: DBLCVT SOURCE FILE         415
HFS: VARCVT SOURCE FILE        2586
HFS: TEST QUOTES SKIDOO         433
HFS: <DIR> BEAGLE BROTHERS
HFS: TEST QUOTES                433
HFS: TEST XTRACT              25600
HFS: TEST ALL                 25600
HFS: COBOL FILE               11900
HFS: COBOL LOG                 1400
HFS: TEST VARIABLE 433
HFS:               11 FILES
HFS:                2 DIRECTORIES
HFS:          100,110 BYTES
HFS:          139,264 BYTES (ACTUAL)
```

“BYTES (ACTUAL)” includes the HFS DASD overhead and will always be larger than the number of data bytes, and is always a multiple of 4096.

Example of FORMAT(2) directory:

```
SYSIN DIR FORMAT(2)
HFS: DIRECTORY OF : ROOT
HFS: 2005/11/23 09:09:11     2,763 VARIABLE TABLE FILE
HFS: 2005/11/28 07:02:41    26,394 AWSTST FILE
HFS: 2005/11/29 11:24:53     <DIR> RENAMED DIR
HFS: 2005/11/30 07:59:28     2,586 VARIABLE CONVERT FILE
HFS: 2005/12/09 09:28:28       415 DBLCVT SOURCE FILE
HFS: 2005/12/15 13:16:39     2,586 VARCVT SOURCE FILE
HFS: 2005/12/22 08:23:56       433 TEST QUOTES SKIDOO
HFS: 2005/12/22 08:34:27     <DIR> BEAGLE BROTHERS
HFS: 2005/12/22 12:42:59       433 TEST QUOTES
HFS: 2005/12/29 08:09:21    25,600 TEST XTRACT
HFS: 2006/01/05 08:34:32    25,600 TEST ALL
HFS: 2006/01/05 08:49:21    11,900 COBOL FILE
HFS: 2006/01/05 08:49:21     1,400 COBOL LOG
HFS: 2006/01/10 11:45:44       433 TEST VARIABLE
```

Here we added file date and time, plus placed the file length in a standard location.

Example of FORMAT(3) directory:

```
SYSIN DIR FORMAT(2)
HFS: DIRECTORY OF : ROOT
HFS:      2,763 VARIABLE TABLE FILE
HFS:     26,394 AWSTST FILE
HFS:      <DIR> RENAMED DIR
HFS:      2,586 VARIABLE CONVERT FILE
HFS:        415 DBLCVT SOURCE FILE
HFS:      2,586 VARCVT SOURCE FILE
HFS:        433 TEST QUOTES SKIDOO
HFS:      <DIR> BEAGLE BROTHERS
HFS:        433 TEST QUOTES
HFS:     25,600 TEST XTRACT
HFS:     25,600 TEST ALL
HFS:     11,900 COBOL FILE
HFS:      1,400 COBOL LOG
HFS:        433 TEST VARIABLE
```

This is essentially FORMAT(2) but dropping the file date and time.

Example of FORMAT(4) directory:

```
SYSIN     DIR FORMAT(4)
  HFS: DIRECTORY OF : ROOT
  HFS: 2006/02/10  07:52:37 <DIR>                      DIRECTORY 001
  HFS: 2006/02/10  07:52:38 <DIR>                      DIRECTORY 002
  HFS: 2006/02/10  07:52:38 <DIR>                      HI THERE
  HFS: 2006/02/28  08:52:08 <DIR>                      JOURNALLED DIR
  HFS: 2006/02/10  07:52:38 <DIR>                      NEW DIRECTORY
  HFS: 2006/02/10  07:52:38          6.9K EBC FIX   80  BIM$DADS.A
  HFS: 2006/02/10  07:52:38        168.5K EBC FIX   80  BIMBMS.A
  HFS: 2006/02/10  07:52:38         31.7K EBC FIX   80  BIMCHDAY.A
  HFS: 2006/02/10  07:52:38          7.9K EBC FIX   80  BIMDYNDS.A
  HFS: 2006/02/10  07:52:38          1.3K EBC FIX   80  BIMFCB.A
       .   .   .
  HFS: 2006/02/15  15:23:06         62.5K EBC VAR       DFHDUMP FILE
  HFS: 2006/02/10  07:52:39         25.0K EBC BIN       LOADED FILE
  HFS: 2006/02/28  09:45:22          1.2M ASC BIN       TEST ASCII FILE
```

Here we sacrificed some accuracy on the file size to be able to include additional information about each file. We show:

EBC – EBCDIC data
ASC – ASCII data

and the file type, which can be:

FIX lll - Fixed length data and record length
VAR - Variable length data
BIN - Binary data, format unknown or unspecified.

FIND

FIND MATCH(‘value’) [FORMAT(3)]

Can be used to locate a specific file, or group of files in the HFS.

MATCH(‘value’)	This parameter is used to select the file or files you wish to locate. The ‘value’ can be generic, using either of: ***** - any number of any characters, or **+** - any single character. For example, “*.HT*” will select every file whose extension begins with “HT” – “HTM”, “HTML”, etc.
FORMAT(3)	There are three possible formats available. These are described in greater detail with the LIST command below. The default is FORMAT(3).

GETCWD

GETCWD

Prints the fully qualified path name of the current working directory. There are no operands for this command.

LIST

LIST [SORT | NOSORT] [FORMAT(3)]

Prints an indented list of the contents of the HFS extent.

SORT | NOSORT	The NOSORT option is retained for compatibility with previous releases of HFS. Using the SORT option provides a better display of the contents of the HFS extent.
FORMAT(3)	There are three formats available. Each is shown in a separate example below. The default is FORMAT(3).

Example of LIST FORMAT(1):

```
SYSIN     LIST FORMAT(1)
<DIR> DIRECTORY 001
      <DIR> SUBDIR 001
            <DIR> SUBSUBDIR 001
                      5.8K JRNL FILE FIVE
                     19.5K JRNL FILE SIX
                5.1K JRNL FILE SEVEN
<DIR> DIRECTORY 002
<DIR> HI THERE
<DIR> JOURNALLED DIR
          2.3K JRNL FILE FOUR
          2.3K JRNL FILE ONE
          2.3K JRNL FILE THREE
          2.3K JRNL FILE TWO
<DIR> NEW DIRECTORY
    6.9K BIM$DADS.A
  168.5K BIMBMS.A
   31.7K BIMCHDAY.A
    7.9K BIMDYNDS.A
    1.3K BIMFCB.A
  145.7K BIMIOGEN.A
   . . .
HBAT:                            7  DIRECTORIES
HBAT:                           79  FILES
HBAT:                    2,933,543  BYTES (DATA)
```

Here the file size is abbreviated to make more room for the file name on a single print line.

Example of LIST FORMAT(2):

```
SYSIN     LIST FORMAT(2)
2006/02/10  07:52:37              <DIR> DIRECTORY 001
2006/02/10  07:52:37                    <DIR> SUBDIR 001
2006/02/10  07:52:37                          <DIR> SUBSUBDIR 001
2006/02/10  07:52:38                                6,000 JRNL FILE
FIVE
2006/02/10  07:52:38                               20,000 JRNL FILE SIX
2006/02/10  07:52:38                          5,280 JRNL FILE SEVEN
2006/02/10  07:52:38              <DIR> DIRECTORY 002
2006/02/10  07:52:38              <DIR> HI THERE
2006/02/28  08:52:08              <DIR> JOURNALLED DIR
2006/02/10  07:52:37                    2,400 JRNL FILE FOUR
2006/02/10  07:52:37                    2,400 JRNL FILE ONE
2006/02/10  07:52:37                    2,400 JRNL FILE THREE
2006/02/10  07:52:37                    2,400 JRNL FILE TWO
2006/02/10  07:52:38              <DIR> NEW DIRECTORY
2006/02/10  07:52:38               7,120 BIM$DADS.A
2006/02/10  07:52:38             172,640 BIMBMS.A
2006/02/10  07:52:38              32,560 BIMCHDAY.A
2006/02/10  07:52:38               8,160 BIMDYNDS.A
2006/02/10  07:52:38               1,360 BIMFCB.A
2006/02/10  07:52:38             149,200 BIMIOGEN.A
```

Here the file date and time are shown, and the file size is accurately listed.

Example of LIST FORMAT(3):

```
SYSIN     LIST SORT FORMAT(3)
               <DIR> DIRECTORY 001
                     <DIR> SUBDIR 001
                           <DIR> SUBSUBDIR 001
                                 6,000 JRNL FILE FIVE
                                20,000 JRNL FILE SIX
                           5,280 JRNL FILE SEVEN
               <DIR> DIRECTORY 002
               <DIR> HI THERE
               <DIR> JOURNALLED DIR
                     2,400 JRNL FILE FOUR
                     2,400 JRNL FILE ONE
                     2,400 JRNL FILE THREE
                     2,400 JRNL FILE TWO
               <DIR> NEW DIRECTORY
               7,120 BIM$DADS.A
             172,640 BIMBMS.A
              32,560 BIMCHDAY.A
               8,160 BIMDYNDS.A
               1,360 BIMFCB.A
             149,200 BIMIOGEN.A
```

LOADFILE

LOADFILE name [REPLACE | NOREPLACE] [DELIM('/+')]

Instructs the batch program to load a file from the card images that follow until a card containing the DELIM(*xx*) in column 1 is encountered.

Name	Specifies the name of the resulting file in the HFS.
REPLACE | NOREPLACE	Governs processing when a duplicate file name is encountered. REPLACE allows overwrites. NOREPLACE will not overwrite the file.
DELIM('/+')	Defines the delimiting characters that will cause the batch program to recognize end of file. The default, "/+," should be sufficient in most cases.

Blank lines will *not* be passed from the parser to HFS, so blank lines cannot be inserted into the file through this mechanism.

You must add an extra line following the trailing delimiter. Normally this is a comment line (beginning with ';'), as shown in the following example:

```
LOADFILE 'LFILE1.TXT'
This file was loaded from cards and should go into
the HFS as fixed-length 80-byte records.
/+
; Extra line required
```

MAKEDIR

MAKEDIR name

Creates a new directory of the name given in the command.

This command can be abbreviated as **MD**.

MOVE

MOVE name1 TO(name2) [REPLACE | NOREPLACE]

Moves a file. The format of the operands of this command are identical to the COPY command described above. The difference between COPY and MOVE is that for MOVE, at end of file, the original file referenced by name1 is deleted from the HFS.

Refer to the COPY command for a description of the operands and name conventions.

OPTION

OPTION [IGNOREDUP | NOIGNOREDUP] – [ALLOWNAME | NOALLOWNAME]

Use this command to change the default behavior of CSIHFBAT.

IGNOREDUP | NOIGNOREDUP	By default, an error is generated whenever you attempt to create a directory that already exists on the HFS. Specify NOIGNOREDUP to bypass these error messages.
ALLOWNAME | NOALLOWNAME	By default, names are edited to disallow certain special characters. This restriction is described above under "File Management." Specify ALLOWNAME to remove these naming restrictions.

READFILE

READFILE name [DUMP | NODUMP] [ASCII | NOASCII] - [DEBUG | NODBUG] [RECFM(format)]

Prints the contents of a file on the HFS.

<table>
<tr><td>Name</td><td>Specify the file name to be printed.</td></tr>
<tr><td>DUMP | NODUMP</td><td>DUMP creates a 32-byte-wide hex-and-character display of the file's contents. DUMP format is assumed when using the RECFM option, see below.

By default, the READFILE command assumes the file is in character with fixed 80 character line lengths.</td></tr>
<tr><td>ASCII | NOASCII</td><td>ASCII causes the character portion of the DUMP output to be translated from EBCDIC to ASCII.</td></tr>
<tr><td>DEBUG | NODEBUG</td><td>DEBUG is intended for use by CSI Technical Support.</td></tr>
<tr><td>RECFM(format)</td><td>Specifies special formatting options for printing.
<table>
<tr><td>RECFM(VAR)</td><td>Request the file be formatted for printing as a standard variable length file, that is with a four byte LLBB preceding each record.</td></tr>
<tr><td>RECFM(FIX len)</td><td>Request the file be formatted for printing as a fixed length file. You must supply the record length with this option.</td></tr>
<tr><td>RECFM(AWS)</td><td>Request the file be formatted for printing as an .AWS file.</td></tr>
<tr><td>RECFM(LF)</td><td>When printed, record-like breaks will occur whenever a CRLF sequence is encountered in the file.</td></tr>
<tr><td>RECFM(HFS)</td><td>The nature of the file is recorded in the HFS for files generated by the HFS File Interception Facility.

This information may not be available for files loaded to the HFS by FTP or some other mechanism.</td></tr>
</table></td></tr>
</table>

REMOVEDIR

REMOVEDIR name

Removes the named directory from the HFS.

This command can be abbreviated as **RD**.

The named directory must be empty, that is, it cannot contain any files of subdirectories. If the directory is not empty, an error will be generated and the command will be rejected.

RENAME

RENAME name1 TO(name2)

Renames a file from its current name to a new name.

name1	Specifies the name of the existing file.
TO(name2)	Specifies the new name for the existing file. By adding the appropriate path information, you can direct the file to a different directory or subdirectory within the HFS extent.

TREE

TREE

This command will print the directory structure, or tree without listing the files contained within each directory or sub-directory. The output is similar to LIST FORMAT(1) as described above.

UPDIR

UPDIR

Moves up one directory level.

This command can be abbreviated as **UD**.

HFS File Journaling

HFS File Journaling provides you with the ability to:

- Journal all HFS update activity for purposes of forward recovery.
- Cache HFS file directories or entire HFS extents to improve performance.
- Create a label area to hold HLBLs (see below) for the HFS File Interception Facility.

Journaling is <u>not</u> available with *HFS-Lite.*

Journaling functions are handled by the program CSIHFJRN, which must run in its own partition. It can be run in either a dynamic or static partition. It can be brought up at any time, but ideally it should be started before the first HFS access in the VSE machine.

Use the following sample JCL as a guide for running CSIHFJRN:

```
* $$ JOB JNM=xxxxxxx,CLASS=x
* $$ LST CLASS=x
// job xxxxxxxx
// DLBL HFSJRNA,'HFS Journal A',0,SD
// EXTENT SYSnnn, . . .                as needed
// DLBL HFSJRNB,'HFS Journal B',0,SD
// EXTENT SYSnnn, . . .                as needed
// LIBDEF                              as needed
// EXEC CSIHFJRN
HFS JOURNAL ( BUFFERS(500) )
HFS CACHE   ( HFSGEN  SIZE( 100 ) CACHEALL )
HFS HLABL   ( 500 )
/*
/&
* $$ EOJ
```

Journal Commands

Journal commands use the same syntax as other batch commands used in HFS and which is described in "Appendix A – Parameter Syntax."

HFS JOURNAL (BUFFERS(256) [TRACE(500)] – [FORCE | NOFORCE])

This command activates Journaling and establishes the number of memory buffers that will be allocated for journal activity.

BUFFERS(256)	Specifies the number of buffers reserved for journal processing. The default is 256.
TRACE(500)	Specifies the number of buffers reserved for HFS diagnostic tracing. The default is 500. Note, this space is not allocated in the Journal partition until a trace command is received from the console. The space is then separately allocated from the partitions 31-bit address space.
FORCE | NOFORCE	In the event that the previous execution of CSIHFJRN failed, either due to program failure or operator cancel, the journal process may have difficulty restarting itself. The FORCE option overcomes this difficulty. **Use this option sparingly, as data loss on the journal file(s) may occur.**

HFS CACHE (filename SIZE(nnn) [CACHEALL | NOCACHEALL])

Use this command to activate caching for an HFS extent.

filename	Specifies the seven character DLBL name for the HFS extent to be cached.
SIZE(nnn)	Specifies the size of the cache in buffers for this file. Cache size is discussed in more detail below.
CACHEALL | NOCACHEALL	Causes HFS to cache all records written to the file as opposed to only directory records. Use this for a control HFS or for an HFS that is heavily hit for read processing, but rarely written to.

HFS HLABL (number)

This reserves label space in 31-bit GETVIS in the Journal partition for use by **INTERCEPTOR** in any of the other partitions in your VSE system.

number	This reserves label space in 31-bit GETVIS in the Journal partition for use by the HFS File Interception Facility. This number represents the maximum labels that are supported by the HFS File Interception Facility at any one time. Since labels can be freed at job step or end of job, this number does not have to be excessively large (see also "Defining HLBLs" below).

HFS Journal Process

Once started, the journal process is automatic. You can message the partition and enter one of the following commands at the VSE console:

CLOSE	Closes the current journal and exits the program
SWITCH	Causes a Journal switch to take place immediately
STATS	Obtain a statistics report on the console (same as close)
TRACE	Generate diagnostic traces, see below.

Once closed, no further journal activity will take place until the journal process is restarted. A program that is currently using HFS may receive warning messages when attempting to modify the HFS file.

No attempt was made to prevent overwriting journal files that are not backed-up. When the program is initiated, it will overwrite the oldest of the two journal files. When journals are switched, the program does so without concern about the state of the other journal file.

Cache Processing

Caching is primarily used with directory records in the HFS file – this can be overridden with the CACHEALL option. Caching will reduce I/Os needed to read records from the HFS. Whenever a record is updated it will still be written to DASD immediately as well as refreshed in the cache.

The HFS cache is maintained in most recently used to least recently used order. Over time, the common portions of the HFS directory (the root directory) will tend to remain at or near the top of the cache. Infrequently used records will migrate deeper into the cache and ultimately be deleted (assuming that the cache is not large enough to contain the entire directory).

You can establish different sized caches for different HFS files. You must supply an HFS CACHE parameter card for each file you wish to cache.

Cache Sizing

It is not necessary to provide a cache sufficient to hold the entire HFS file directory. Because the cache is organized in most recently used to least recently used order, the common elements of the file will tend to remain in the cache and near the top of the cache index. Use the LIST and the STATS command (see "HFS File Management") to help determine the most effective cache size.

HFS Trace

HFS tracing is primarily intended for the use of CSI Technical Support as a diagnostic tool to assist in problem resolution. It is controlled through the console interface to CSIHFSJRN. Tracing recognizes the following four commands:

TRACE	Queries the current trace status.
TRACE ON	Starts the trace. All subsequent HFS activity will be traced.
TRACE OFF	Stops the trace.
TRACE PRINT	Prints the contents of the trace buffers to the POWER LST queue to job named "HFSTRACE" which is placed into class L, disp H.

Sample trace report:

```
H F S T R A C E
07.26.10 F5 004CF200 00:OPEN             0000000000 RET=00 FILE=CSIHFDT
07.26.10 F5 004CF200 02:CHANGE DIRECTORY 0000000000 RET=00
FN=/
07.26.10 F5 004CF200 07:DIRECTORY        00B0000000 RET=00
FN=/
07.26.10 F5 004D2480 00:OPEN             0000000000 RET=00 FILE=CSIHFDT
07.26.10 F5 004D2480 02:CHANGE DIRECTORY 0000000000 RET=00
FN=/DIRECTORY ONE
07.26.10 F5 004D2480 07:DIRECTORY        00B0000000 RET=00
FN=/DIRECTORY ONE
07.26.10 F5 004D2480 07:DIRECTORY        0070000000 RET=00
FN=/DIRECTORY ONE
07.26.10 F5 004D2480 07:DIRECTORY        0070000000 RET=00
FN=/DIRECTORY ONE
07.26.10 F5 004D2480 07:DIRECTORY        0070000000 RET=00
FN=/DIRECTORY ONE
07.26.10 F5 004D2480 07:DIRECTORY        0070000000 RET=04
FN=/DIRECTORY ONE
07.26.10 F5 004D2480 01:CLOSE            0070000000 RET=00
FN=/DIRECTORY ONE
07.26.10 F5 004CF200 07:DIRECTORY        0070000000 RET=00
FN=/
07.26.10 F5 004D2480 00:OPEN             0000000000 RET=00 FILE=CSIHFDT
07.26.10 F5 004D2480 02:CHANGE DIRECTORY 0000000000 RET=00
FN=/DIRECTORY TWO
```

Journal Management

Two journal files are used, HFSJRNA and HFSJRNB. These are sequential disk files and processed throughout the journal process by standard VSE LIOCS using DTFSDs. These files need to be initialized before they are first used.

When starting up, CSIHFJRN chooses between the two journal files by opening both and selecting the oldest one, that is the one with the least recent date and time stamp. Its choice is displayed on the system console during start up.

When a journal file fills up it is automatically switched with its counterpart. This switch must occur and no attempt is made to determine if the other file has been backed up. You can also manually cause the switch to occur through the console interface as described above.

You can use any utility program you wish to backup the journal file, but due to processing considerations unique to the journaling environment, CSI strong recommends that you use CSIHFBAT and the Journal Backup function described below for backing up the journal files to tape.

Ideally the journal file will contain a normal end-of-file marker. However, with a long running operation like journaling, this will not always be the case. An abnormal termination of journaling, such as a forced IPL of the VSE machine, may leave the journal file without its end-of-file marker. The HFS Journal Backup process is aware that this can happen and determines end-of-file to be any of:

- Normal end-of-file.
- An I/O error of any kind.
- A break in the sequence of date and time stamps in the individual journal records. (Because it is possible, but not likely, that under extreme stress the journal records can get out of strict time order, this test is necessarily a bit fuzzy.)

Journal Management Commands

HFS Journal Management Commands consist of three functions, each of which is performed by CSIHFBAT. They are documented separately here in the attempt to avoid confusion. See “HFS File Management” above for more information about CSIHFBAT. See “Appendix !- Parameter Syntax” for information on the syntax and documentation conventions for these commands.

JRNL INIT

Initializes both journal files: HFSJRNA and HFSJRNB. DLBL and EXTENT information for both files must be available to the job.

The ACCESS command is not needed for this process.

JRNL BACKUP(filename)

Use this command to backup a journal file to tape. The tape backup is written to file named "JRNBKUP" on SYS009. The ACCESS command is not needed.

filename	Can be either "HFSJRNA" or "HFSJRNB."

Use the following JCL as a guide for the backup process:

```
. . .
// ASSGN SYS019,DISK,VOL=SYSWK1,SHR
// DLBL HFSJRNA,'HFS JOURNAL A',0,SD
// EXTENT SYS019,SYSWK1
// DLBL HFSJRNB,'HFS JOURNAL B',0,SD
// EXTENT SYS019,SYSWK1
// ASSGN SYS009,500
// MTC REW,SYS009
// TLBL JRNBKUP,'JRNBKUP'
// LIBDEF *,SEARCH=as needed
// EXEC CSIHFBAT,SIZE=AUTO
JRNL BACKUP(HFSJRNA)
/*
. . .
```

In order for a subsequent JRNL RECOVER to function properly, you must take periodic backups of each HFS you wish to recover (See "HFS File Management" above for information on the BACKUP command.) In addition you must ensure that:

1. **You must switch the journal proper to creating the backup, and**
2. **You must use the PHYSICAL option of the BACKUP command.**

If these steps are not done in this order the results of the JRNL RECOVER process are unpredictable and you will experience data loss.

JRNL RECOVER

Use this command to recover an HFS using backed up journal files. The journal files are read from a tape named "JRNREST" using SYS009. Unlike the other JRNL commands, here the ACCESS command is relevant and indicates the HFS extent to be recovered.

Use the following JCL as a guide for the restore process.

```
. . .
// ASSGN SYS019,DISK,VOL=SYSWK1,SHR
// DLBL HFSJRNA,'HFS JOURNAL A',0,SD
// EXTENT SYS019,SYSWK1
// DLBL HFSJRNB,'HFS JOURNAL B',0,SD
// EXTENT SYS019,SYSWK1
// ASSGN SYS009,500
// MTC REW,SYS009
// TLBL JRNREST,'JRNBKUP'
// LIBDEF *,SEARCH=as needed
// EXEC CSIHFBAT,SIZE=AUTO
ACCESS HFS01
JRNL RECOVER
/*
. . .
```

If it becomes necessary to do a forward recovery using journaled data, you must

1. **Ensure you have a Journal Backup for the most recent activity to the affected HFS file.**
2. **Initialize the affected HFS (see "HFS File Management" for instruction on the INITIALIZE command).**
3. **Restore the most recent PHYSICAL backup tape.**
4. **Pass Journal Backup data through the JRNL RECOVER process in the least recent to most recent order. Your tape manager should allow you to concatenate multiple tape files together into a single execution of the JRNL RECOVER process.**

Failure to follow these steps will cause unpredictable results and data loss.

Irregardless of the procedures used, any HFS activity that was in process when the failure occurred (that forced you into using the JOURNAL RECOVER process) will likely be lost. The JRNL RECOVER process cannot resume an in-flight task.

Internally, the JOURNNAL RECOVER process runs a DEEP RECOVERY to clean up any stray records and partially built files. When recovery is complete, the HFS EXTENT should be intact and contain all file activity that was completed prior to the original failure.

HFS File Interception Facility

When the HFS File Interception Facility is active, HFS examines each open of a DTF. It checks to see if this is an open of a DTFSD and then scans its internal HLBL table for a matching file name. If everything matches up, HFS replaces the standard IBM I/O module with the HFS I/O module (CSIHLMOD) and all subsequent I/O activity is under the control of HFS.

This process is completely transparent to you. Your programs do not need to be aware that this is happening – in fact, they cannot detect it. HFS uses the DTF in the same manner as IBM LIOCS, obtaining data from the DTF and setting information back into the DTF in the same manner as standard VSE LIOCS.

Once it has taken over your I/O activity, HFS is able to provide many services hitherto unavailable to you.

- Writes data to and retrieves data from an HFS
- Optionally, HFS can encrypt and decrypt this data as needed
- Optionally, HFS can convert the data into a variety of different file formats.

And you don't have to do a thing to your programs to get this to happen, a simple JCL change makes it all work.

The remainder of this section describes this JCL change in detail. It's really not as complex as it looks as the HLBL, the control statement that defines the file to the HFS File Interception Facility, was modeled after the standard VSE DLBL and the usage for many of the operands will be obvious to you.

File Interception is <u>not</u> available with *HFS-Lite.*

Defining HLBLs

For each file that you want the HFS File Interception Facility to handle you need to supply an **HLBL** in the JCL. The **HLBL** itself is intentionally constructed similar to a DLBL to make it easier to use and remember.

There are additional commands available to simplify and control the **HLBL** process.

All of these commands are processed by program CSIHLABL which will need to be inserted into your JCL ahead of your program which generates the file. This JCL looks like:

```
// LIBDEF *,SEARCH=(                    as needed
// EXEC CSIHLABL,SIZE=AUTO
  . . .  commands here . . .
/*
```

See "Appendix A – Parameter Syntax" for notes on command syntax.

CSIHLABL Commands

OPTION

The **OPTION** command is itself optional and for the most part needed only if you must use something other than the defaults.

OPTION [ADD | NOADD] [**CLEAR**] [SEP(sep)]

ADD | **NOADD**	By default, whenever CSIHLABL is executed, HFS labels for this partition are removed and replaced by the **HLBL**(s) that follow. If you want to add a label to those already established for the partition then specify OPTION ADD

CLEAR	HFS labels will remain available in the partition until either an IPL, a shutdown of the HFS Journaling partition, or a subsequent execution of CSIHLABL. This could result in a potential security breach with encrypted data.
	
	To close the security hole, you should add a final execution of CSIHLABL to the end of your job with OPTION CLEAR to wipe out the labels and making that space available for other partitions. This step should be add as a separate VSE JOB in the JECL like:
	
	`/&`
	`// JOB CLEAR`
	`* CLEAR OUT INTERCEPTOR HLBLS`
	`// EXEC CSIHLABL,SIZE=AUTO`
	`OPTION CLEAR`
	`/*`
	`/&`
	`* $$ EOJ`
	
	This step will get executed even if the job is cancelled, or an abend occurs.

SEP (sep)	By default the directory separator is a forward slash, "/". You can change this value to anything you want by entering the separator character here. Quotes are not necessary. You cannot use a period, any other character on your keyboard is available for use. For example, SEP(\) will change the separator to the back slash used by Microsoft™ on the PC. It is up to you to ensure that the separator character does not appear in any of the file names as other than a separator.

HLBL

The HLBL command defines a file for use by HFS File Interception Facility.

```
HLBL dlbl.name                                  -
     hfs.file.name                              -
     retention                                  -
     file type                                  -
     HFS (hfs.extent)                           -
     DISP (current normal abnormal)             -
     [ SECURE ( algorithm key.number) ]         -
     [ CONVERT (script.name) ]                  -
     [ CRLF | NOCRLF ]                          -
     [ EOF | NOEOF ]                            -
     [ AWS | NOAWS ]                            -
     [ APPEND | NOAPPEND ]                      -
     [ RENAME | NORENAME ]                      -
     [ UNDEF | NOUNDEF ]
```

dbl.name	This is the seven character file name (or DLBL name) that your program will use when attempting to access the file. This name must be supplied.
hfs.file.name	This is the name of the file in the HFS extent. This must be the fully qualified path name for the file beginning with a separator character. This name must be supplied. The name can contain one of the two special keywords. %DATE – results in today's date being placed in the label %TIME – results places the current time in the label. **NOTE:** this is the date and/or time that the label was generated by the CSIHLABL program, not the exact date and time that the file was created. This was done in this manner to allow step 1 to write the file and step 2 to read it without having to fiddle with the labels in between. For an input file, the name can be generic. HFS will select all files that match the generic name in the indicated directory, sort them, and return them one at a time to the input DTFSD. Generic names can use: ***** - any number of any characters, or **+** - any single character.

<table>
<tr><td>Retention</td><td>This can be either a number of days of retention or it can be entered in the form yy/nnn which identifies a specific day and year. This is much the same as available on the VSE DLBL statement. If an attempt is made to overwrite an unexpired file, the operator will be prompted with a request to continue the job or cancel it in much the same manner as VSE does for normal files. (See Messages for CSIHLMOD below.) Retention must be supplied.</td></tr>
<tr><td>file type</td><td>This is two characters and should be either “SD” or “MT.” In this release of HFS this field is not edited but must be supplied. Future releases of HFS may take advantage of this field so it is best to match the dataset organization of the file as you would normally do.</td></tr>
<tr><td>HFS (hfs.extent)</td><td>You must supply the DLBL name of the HFS extent that is to contain this file. This operand must be present unless the HLBL is preceded by an HEXT command (see below).</td></tr>
<tr><td>DISP(current normal abnormal)</td><td>Use the DISP option to control the file processing for normal and/or abnormal terminations. There are three components to this parameter:<table><tr><td>Current</td><td>Can be OLD or NEW</td></tr><tr><td>Normal</td><td>Determines how to treat the file when the job step completes normally. This can be either KEEP or DELETE.</td></tr><tr><td>Abnormal</td><td>Determines how to treat the file when the job step abnormally terminates. This can be either KEEP or Delete.</td></tr></table>For example, DISP(OLD DELETE KEEP) indicates that the file should currently exist, it will be deleted upon normal completion and kept if there is a program failure of some kind.</td></tr>
<tr><td>SECURE(algorithm key)</td><td>The SECURE operand causes HFS to encrypt the file using specifications supplied in the operand. Encryption is somewhat complex and is described separately below.</td></tr>
<tr><td>CONVERT(script)</td><td>Use this operand to supply the name of a separately constructed conversion script. The name can be up to 248 characters in length and must not begin with the separator character. The file conversion process is non-trivial and described separately below.</td></tr>
<tr><td>CRLF | NOCRLF</td><td>HFS can provide some assistance for your programs when dealing with CRLF delimited files directly. This is discussed in greater detail in “Appendix C – CRLF Processing” below.</td></tr>
</table>

EOF | **NOEOF**	By default when an input file is not found in its associated HFS extent, HFS will cancel the job with a "File Not Found" message. If you specify EOF, HFS will instead return an immediate end-of-file indication to your program in response to the first read attempt.
AWS | **NOAWS**	Specify AWS to cause HFS to generate a file formatted as an .AWS tape. AWS formatted developed according to specifications at www.bustech.com/support/techtips/mas/awstape.htm.
RENAME | **NORENAME**	This option applies to an output file only. If RENAME is specified and a duplicate file is encountered in the HFS extent, the file name will be examined, from right to left, for the number string "000." If found, the three digit number will be incremented and the writer attempt re-tried. The number string can be located anywhere within the file name. For example, given a file name of "/TEST DATA 000.TXT" The first file will be written as "/TEST DATA 000.TXT." A subsequent file will be "/TEST DATA 001.TXT," and so on.
APPEND | **NOAPPEND**	This applies only to an output file and if APPEND is specified indicates that the output file should be added to an existing file in the HFS. If no file exists, anew file will be created.
UNDEF | **NOUNDEF**	Use of DTFSD RECFORM=UNDEF files in assembler language programs present problems to HFS. This difficulty is described in detail in "Appendix B – Undefined Files" below.

HEXT

This command allows you to specify the HFS extent for use with all subsequent HLBLs. The hfs.name is the seven character DLBL name of the HFS extent. Any subsequent HLBL that does not contain the "HFS(name)" operand will use the HFS specified here.

HEXT hfs.name

See the Examples below for samples of this command's usage,

HSET

The HFS directory given in this command applies to all subsequent HLBL commands. If an HLBL contains ah HFS name that does not begin with a separator character, then the actual name used will be constructed by using the directory name given here followed by the file name in the HLBL.

HSET hfs.directory

This will be clearer after reviewing the Examples below.

LIST

This command lists the labels currently known to the HFS File Interception Facility.

LIST [PARTITION(ALL | id)]

If the PARTITION operand is omitted, only those labels for the partition that is executing CSIHLABL will be shown.

If PARTITION is specified it can be either:

ALL - for all labels known to HFS, or
Id - for labels for the associated partition (BG, F1, F2, etc.)

The LIST command will not cause the label area to be cleared for this partition (see above).

REMOVE

This command will remove a single file from the labels for this partition.

REMOVE file.name [PARTITION(id)]

If the PARTITION operand is omitted, the label will be removed from the current partition.

If PARTITION is specified, the id must be the partition id (BG, F1, F2, etc.) where the removal is to take place.

CSIHLABL – Examples

1. Simple Case

```
// EXEC CSIHLABL,SIZE=AUTO
 HLBL FILE1 '/TEST FILES/FILE1' 0 SD HFS=HFS01
 HLBL FILE2 '/TEST FILES/FILE02.AWS' 0 SD HFS=HFS01 AWS
 HLBL FILE3 '/SECURE/FILE03' 3 SD HFS=HFS02 –
            SECURE(AES128C-SHA1 2)
/*
```

FILE1 will be directed to HFS01 into a file named '/TEST FILES/FILE1'. It will be written in EBCDIC in the clear.

FILE2 will also be directed to HFS01 into a file named '/TEST FILES/FILE92.AWS.' It will be created in AWS format.

FILE3 will be directed to HFS02 into a file named '/SECURE/FILE03'. It will be encrypted using the AWS128C-SHA1 algorithm and key 2 in the user defined keys.

2. Using HEXT

```
// EXEC CSIHLABL,SIZE=AUTO
 HEXT HFS01
 HLBL FILE1 '/TEST FILES/FILE01.SAMP' 1 SD
 HLBL FILE2 '/SPLISH/SPLASH/BATH' 0 SD   -
            CONVERT('TOWEL EM OFF')
 HLBL FILE3 '/SECURE/FILE03' 3 SD HFS=HFS02 –
            SECURE(AES128C-SHA1 2)
/*
```

FILE1 will be directed to HFS01 (from the HEXT command) into a file named "/TEST FILES/FILE01.SAMP."

FILE2 will likewise be directed to HFS01 into file named "/SPLISH/SPLASH/BATH." It will be converted according to the rules described in "TOWEL EM OFF."

FILE3 will be directed to HFS02 (the HFS= in the HLBL overrides the HEXT setting) into a file named '/SECURE/FILE03'. It will be encrypted using the AWS128C-SHA1 algorithm and key 2 in the user defined keys.

3. Using HSET

```
// EXEC CSIHLABL,SIZE=AUTO
 HSET '/TEST FILES'
 HLBL FILE1 'FILE01.SAMP' 1 SD HFS=HFS01
 HLBL FILE2 '/SPLISH/SPLASH/BATH' 0 SD HFS(HFS01)  -
            CONVERT('TOWEL EM OFF')
/*
```

FILE1 will be directed to HFS01 into a file named "/TEST FILES/FILE01.SAMP" – the directory specified in the HSET is prefixed to the file name.

FILE2 will likewise be directed to HFS01 into file named "/SPLISH/SPLASH/BATH." Since the HFS name in the HLBL began with the directory separator, the directory specified in the HSET command is ignored. It will be converted according to the rules described in "TOWEL EM OFF."

HSET and HEXT can be used together in the same execution of CSIHLABL. You can also use multiple HSET and HEXT commands, their settings apply to all subsequent HLBL commands.

File Encryption

HFS provides several different methods of encryption each of which can be further distinguished by user-defined keys. Encryption is requested by adding the **SECURE** operand to the **HLBL** command (see "Defining HLBLs" above). The SECURE operand is described below:

File encryption is <u>not</u> available with *HFS-Lite.*

HLBL SECURE operand

SECURE (algorithm key.number)

There are two options for the SECURE operand, algorithm and key number.

Algorithm can be one of:

SDESCBC-NULL	Data Encryption Standard (DES).
SDESCBC-SHA1	Data Encryption Standard (DES). Includes **SHA1** Secure Hash Algorithm in the encryption.
TDESCBC-NULL	Triple Data Encryption Standard.
TDESCBC-SHA1	Triple Data Encryption Standard. Includes **SHA1** Secure Hash Algorithm in the encryption.
AES128C-NULL	Advanced Encryption Standard (AES).
AES128C-SHA1	Advanced Encryption Standard (AES). Includes **SHA1** Secure Hash Algorithm in the encryption.
DFLT	Your installation default encryption algorithm. See "Establishing Keys" below for more information.

Details of the various encryption algorithms can be located on the web. (To start, Google™ "DES", "TDES", "AES128" and/or "SHA1" and go from there.)

The **Key Number** can be any value from 1 to 9999 or the keyword "DFLT". You <u>must</u> have a corresponding key value in the table discussed in the next section.

Establishing Keys

HFS is shipped with three different keys, numbered 1, 2 and 3. These are the same keys that everybody who installs HFS or *TCP/IP for VSE* has upon installation. If this bothers you, these key values can be customized.

For security reasons, the customization process is not documented here. Contact CSI for instructions on how to make these keys unique for your installation.

Security Considerations

1. Performance

There's no getting around it, performance will be not be good on older IBM hardware. Encryption is not free and involves a substantial amount of CPU overhead. If you are running on one of the newer Z-series machines, CSI will take advantage of the hardware CP Assist for Cryptographic Functions (CPACF) and performance is significantly improved – contact CSI Technical Support for instructions on how to activate the hardware support.

2. Securing the Key Values

When an HLBL is encountered, the command is printed on SYSLST as part of the execution of program CSIHLABL. The program looks to see if the SECURE operand is included, and if so, its contents are suppressed,. For instance, if your HLBL looks like:

HLBL file1 '/file on1' 3 SD HFS-HFS01 SECURE(AES128C-SHA1 2)

It will be printed as:

HLBL file1 '/file on1' 3 SD HFS-HFS01 SECURE(**********)

In this manner, anyone viewing the listing knows that the file has been encrypted but not how.

This still leaves the actual algorithm and key values in the clear in the JCL in the POWER RDR queue, and in the text editor that you use to create and maintain the JCL. Your text editor should have provisions to allow you to secure the contents of the JCL from prying eyes.

You probably do not want to leave the JCL in the POWER RDR queue in DISP=L for the simple reason that anyone who knows how to look at the RDR queue using your online viewer (such as ICCF, BIM-EDIT, RAAD, etc.) can easily discover the encryption methodology.

Even so, modern third party viewers can look at the JCL as it is executing so some vulnerability still exists. Check with your vendor. RAAD, available from CSI, can be restricted by individual job name and queue to clamp down on unauthorized access.

File Conversion

HFS File Interception can be used to convert data formats when writing and reading files to an HFS.

- You can easily convert mainframe records in to comma separated CRLF delimited strings for input to programs on external systems.
- These comma separated CRLF delimited strings can in turn be read by HFS File Interception and converted back to mainframe data formats (some restrictions apply, see below).
- The separator does not have to be a comma, any single character can be used.
- Packed decimal and binary fields will be converted properly, with or without a decimal point.
- File conversion can result in HTML and/or XML output as well.

File conversion is <u>not</u> available with *HFS-Lite.*

File conversion is activated by adding the CONVERT operand to the HLBL statement as described previously. Before you can do that, however, you need to supply:

1. File Definition
2. Conversion Rules.

Both the File Definition and Conversion Rules are processed by program CSIHFSDX. The program's output is directed to the HFSGEN file described in the "System Requirements" section of this manual. Use the following JCL as a guide for CSIHFSDX.

```
// DLBL HFSGEN,…          as needed
// EXTENT SYS009,…        as needed
// ASSGN SYS009,…         as needed
// LIBDEF *,SEARCH=…      as needed
// EXEC CSIHFSDX
 csihfsdx commands are placed here
 . . .
/*
```

A File Definition is processed from encountering a **FILE** command (see below) and continues until the **END** command is encountered. Likewise the Conversion Rules are processed from the **CONVERT** command to the ensuing **END** command. Obviously the **END** is present for both. In addition, both File Definition and Conversion Rules contain a **TABLE** command whose requirements vary slightly depending on whether it is found within the Conversion Rules or a File Definition.

Multiple sets of Conversion Rules can be created referencing a single File Definition.

In addition to creating the definitions, CSIHFSDX can also print them out for your review. This is done with the **LIST** command.

CSIHFSDX also handles user-supplied translate tables for File Conversion. In the possible misguided belief that this will happen very rarely, the description of the TRANSLATE command was moved to "Appendix D – Translate Tables" so as to not clutter up the manual at this point.

General Commands

There are two commands which fall outside of the File Definition and/or Conversion Rules processes.

OPTION

OPTION [WARNINGS | NOWARNINGS]

CSIHFSDX can issue several warning messages during processing. You can suppress these messages by specifying NOWARNINGS.

LIST

LIST {FILE | CONVERT} (file name)

This command will list the contents of the named File Definition or Conversion Rules file in a columnar format for easy reference.

LIST FILE(name) will list the named File Definition.

LIST CONVERT(name) will list the named Conversion Rules.

File Definition

Before conversion can take place a file definition must be established. You should be able to create this easily from an existing COBOL file description.

You can provide multiple File Definitions in a single execution of CSIHFSDX if you want to.

FILE

FILE ‘name’ {RECFM(VAR max) | RECFM(FIX lrecl)}

The **FILE** command begins the File Definition process and provides a name for the File Definition. Its operands are:

‘name’	The file name can be from 1 to 248 characters in length and must begin with a non-blank character. It must not include and embedded slash ‘/’. This name will be referenced by a subsequent set of Conversion Rules. The name should be enclosed in single quotes as shown. If a definition already exist of this name it will be deleted and replaced by the definition formed from the commands that follow the FILE command.
RECVM(VAR max)	If the file is a standard variable file, specify RECFM(VAR). You must also supply the maximum record size.
RECVM(FIX lrecl)	If the file is a fixed length file, then you need to specify it as FIX and also supply the logical record length (lrecl).

FIELD

FIELD offset name -
[PIC('cobol.pic' cobol.usage) |
FMT(length type decimals signed)]

Use the **FIELD** command to define each field in the file.

You will probably find it easiest to define the fields using the COBOL-like PIC operand. However, when printed using the **LIST** command (above) the FMT option will be displayed- no attempt was made to preserve or reformulate the PIC.

Offset	This is the offset into the record for this field. The offset begins from zero. A special offset of asterisk ('*') is available and indicates that CSIHFSDX is to determine the offset for this field based on the lengths of the fields preceding the FIELD command. The first FIELD command for the File Definition can be an asterisk. (See also the discussion on "Offsets" below for more information.)
Name	You need to supply a unique name for the field. The COBOL name will probably suffice. The name can be from 1 to 30 characters in length and must begin with a non-blank character. The COBOL keyword "FILLER" may be used.
PIC(pic usage)	CSIHFSDX supports a subset of the range of possible COBOL PICs. This is described in more detail below. Unlike COBOL, however, you will need to enclose the PIC pattern in single quotes.
FMT(length type decimals signed)	If you prefer, you can explicitly define the field by supplying its Length Type – can be ZD, PD, CH or BI Number of decimals SIGNED | NOSIGNED for numerics

COBOL PIC support is provided for the most commonly used PICs. Edited PICs are not supported. Usage COMP and COMP3 are also supported. For example:

PIC	LENGTH	RESULT
X	1	One byte character
XXX	3	Three byte character
X(30)	30	30 byte character
9999	4	4 byte Zoned Decimal (ZD)
S9999	4	4 byte signed Zoned Decimal
S99999 COMP-3	3	3 byte Packed Decimal (PD)
S9(9)V99 COMP-3	6	6 byte Packed Decimal, two decimal places identified
S9 COMP	2	2 byte binary (half-word)
S9(4) COMP	2	2 byte binary (half-word)
S9(5) COMP	4	4 byte binary (full-word)
S9(9) COMP	4	4 byte binary (full-word)
S9(10) COMP	8	8 byte binary (double-word)
S9(18) COMP	8	8 byte binary (double-word)

TABLE

TABLE offset name OCCURS(nnn) DEPENDSON(name)

This defines a table within the record.

Tables complicate things somewhat and the rules for offsets are different when tables are present. Read the discussion on "Offsets" below carefully when dealing with tables in your records.

Offset	Offset is specified in a similar manner to the FIELD command described above. (See also the discussion on "Offsets" below for more information.) The end of a table is indicated by the special offset "END", like: TABLE END name. The name supplied with TABLE END must be the name provided in the corresponding initial TABLE command for the table.
Name	A unique name must be supplied for the table. The name is formulated according to the same rules given above for FIELD.
OCCURS(nnn)	You must supply a OCCURS count for the table. If this is an OCCURS DEPENDING ON situation, you must supply the maximum number of table occurrences.
DEPENDSON(name)	The name supplied here is the name given to a previously defined FIELD which must be numeric and contains the number of table entries in this record.

All FIELD commands encountered subsequent to a TABLE command are considered to be part of the table itself until a TABLE END is encountered. You must have a TABLE END for each TABLE you define. In addition the offsets for the FIELDs with the table restart at zero as they are the offset from the beginning of the table, not the beginning of the file – this restriction is also covered in the discussion on "Offsets" below.

TABLE commands can be nested, that is a TABLE can be present within the scope of a preceding TABLE command. Care must be taken in providing offsets in this case – see "Offsets" below. Both tables need to be terminated with a corresponding TABLE END command.

END

END

The END command is required and terminates the File Definition started by a preceding FILE command.

When the END command is encountered, the preceding File Definition is verified to ensure that all FIELD names are unique, that the record length matches the calculated length for fixed length records, all TABLE commands are properly terminated, and that any DEPNDSON entries exist in the file and are of the proper format.

You must clean up the verification errors, if any, before attempting to use the File Definition. The actual conversion processor assumes that the File Definition is correct and bad things will happen if an erroneous File Definition is used.

Examples

1. Simple Fixed Length Record

```
FILE 'FIXED TEST FILE 1' RECFM(FIX 50)
 FIELD   * RCD-KEY              PIC('9999')
 FIELD   * RCD-NAME             PIC('X(30)')
 FIELD   * RCD-TRANSACTION      PIC('S9(9)' COMP-3)
 FIELD   * RCD-SSN              PIC('S9(9)' COMP-3)
 FIELD   * RCD-TRAN-TYPE        PIC('X')
 FIELD   * RCD-TRAN-AMT         PIC('S9(7)V99' COMP-3)
 END
```

This is a simple fixed length record illustrating the various PICs that are supported. An asterisk was used for all offsets.

2. Fixed Length Record and TABLE

```
FILE 'FIXED TEST FILE 2' RECFM(FIX 112)
 FIELD   * RCD-KEY              PIC('9999')
 FIELD   * RCD-NAME             PIC('X(30)')
 TABLE * TBL_MONTHLY OCCURS(12)
 FIELD   0 RCD-TRAN-TYPE        PIC('X')
 FIELD   * RCD-TRAN-AMT         PIC('S9(7)V99' COMP-3)
 TABLE END TBL_MONTHLY
 FIELD 106 RCD-TOTAL            PIC(‘S9(7)V99’ COMP-3)
 FIELD   * RCD_FLAG             PIC X
 END
```

Here we introduced a 12 entry table in the midst of the record. Note the offsets which are discussed in more detail below.

3. Variable Length Record

```
FILE 'VARIABLE TEST FILE 1' RECFM(VAR 5532)
 FIELD   * RCD-KEY       PIC('9(9)')
 FIELD   * RCD-NAME      PIC('X(21)')
 FIELD   * RCD-ENTRIES   PIC('S9(4)' COMP)
 TABLE   * RCD-ACTION    OCCURS(100)  DEPENDSON(RCD-ENTRIES)
 FIELD   * TBL-TIME      PIC('9999')
 FIELD   * TBL-AMPM      PIC('X')
 FIELD   * TBL-ACTION    PIC('X(50)')
 TABLE END RCD-ACTION
 END
```

This example illustrates the COBOL OCCURS DEPENDON ON situation.

Offsets

For the most part, CSIHFSDX is able to determine the proper offset to individual fields by itself. Calculations are done correctly in both examples 1 and 3 above.

Automatic calculation of offsets will work properly provided that all fields are defined with corresponding FIELD commands. CSIHFSDX supports the COBOL keyword "FILLER" in that it is not checked for duplicate file names upon verification.

Automatic calculation may not work correctly if the record contains a table. It will function correctly if the table is the last thing in the record as illustrated in example 3 above. It will fail, however, if there is non-tabular data following the table as shown in example 2 above. Here, you will need to supply the actual offset, relative to zero, of the first field following the TABLE END in order to reset the internal offset calculations. This was done in example 2 above.

It is critically important that all fields within a table are defined in the File Definition. CSIHFSDX determines table length depending on the aggregate length of the FIELDS plus coded offsets found within the TABLE boundaries.

Nested tables should work fine unless the inner table is also the last thing within an outer table. For example (in COBOL syntax)

```
10 TABLE-1  OCCURS 10
  15 FIELD  PIC…
  15 TABLE-2 OCCURS 10
    20 FIELD  PIC…
10 FIELD_NOT_IN_TABLE  PIC…
```

CSIHFSDX will have trouble determining the length of TABLE-1 by itself. If you have a record of this nature, contact CSI Technical Support for assistance.

Conversion Rules

In its simplest case a set of Conversion Rules is a list of fields that are to be moved from the previously defined File Definition to the output file. Not all fields that are defined have to be moved, nor do they have to be moved in the same order as defined.

Conversion normally implies that numeric data in mainframe format (packed decimal or binary) will be converted to a character string when moved to the output file. Likewise, when converting from a CRLF delimited string to mainframe format, the character representation of numeric data will be converted into the proper mainframe format.

Conversion processing can also create UNIX-like binary files. This involves special handling for both character and numeric data.

A rudimentary logic capability along with the ability to insert data at definable points expands the File Conversion beyond simple comma separated, CRLF delimited strings into a variety of output formats.

The same set of Conversion Rules is used to convert data from or to mainframe format. Note, however, that comma separated CRLF delimited strings and LINUX-like binary files are the only formats that can be used to convert character based data back into mainframe formats.

The direction conversion takes depends on how the sequential file is opened. Files opened for output will cause conversion of mainframe data to string data to take place. Files opened for input will cause conversion of string data to mainframe format to occur.

CONVERT

CONVERT ‘name’ ‘definition’ [ASCII | **NOASCII] [**SEP(sep)] –
[NULL(value)] [DECIMAL(value)] –
[TRANSLATE(scope ‘name’)]

The CONVERT command begins the definition of a set of Conversion Rules. It provides a unique name for this set of rules and identifies the file definition that describes the data.

‘name’	The file name can be from 1 to 248 characters in length and must begin with a non-blank character. It must not include and embedded slash ‘/’. The name should be enclosed in single quotes as shown. If a set of Conversion Rules already exist of this name it will be deleted and replaced by the rules formed from the commands that follow the CONVERT command.
‘definition’	This is the name of a previously defined File Definition. The File Definition must exist prior to issuing the CONVERT command. The definition name can be from 1 to 248 characters in length and must begin with a non-blank character. It must not include and embedded slash ‘/’. The name should be enclosed in single quotes as shown.
ASCII|**NOASCII**	Optional parameter. Specify ASCII to cause CSIHFSDX to translate the output string to standard ASCII format. The default is NOASCII which indicates no translation.
SEP(sep)	Optional parameter. This provides the field separator to be used when moving data to the file. The default is a comma.
NULL(value)	Optional parameter, relevant to string to mainframe conversion only. The value entered here will be used to fill the mainframe record prior to any data movement. The default is binary zeroes (x’00’). You can enter any character value here. To change this to a space (x’40’) use the keyword “SPACE”.
DECIMAL(value)	Optional parameter. Provides the character used to indicate decimal position in the character numeric format. Currently there are only two choices: “PERIOD” or “COMMA.” If this is insufficient, contact CSI Technical support for assistance. The default is a period.

TRANSLATE(scope 'name')	Optional parameter. If present data will be translated according to the specification in the named table. The "scope" can be either "ALL" or "NONE". "ALL" indicates that each output line is to be translated accoriding to the named table. "NONE" indicates that translation will be determined on individual MOVE commands (see below). The table name can be the keyword "ASCII" which indicates that the internal EBCDIC to ASCII and ASCII to EBCDIC tables should be used for the translation. Otherwise, the name refers to a translate table created using the procedures discussed in "Appendix D – Translate Tables." (A set of code page translation tables are available separately from CSI.)

MOVE

MOVE name [TRUNC | NOTRUNC] [SEP | NOSEP] -
[UPPER | NOUPPER] [LOWER | NOLOWER] –
[TRANSLATE | NOTRANSLATE] -
[DECIMAL | NODECIMAL] –
[BINARY(BIG|LITTLE)] -
[LENGTH(length)] -
[QUOTE (SINGLE | DOUBLE)] –
[EDIT (pattern)]

This command causes data to be transferred in either direction.

Name	This is the field name used in the associated File Definition for this conversion process, which is, of course, the field that will be moved. This is the only operand that is required, all remaining operands are optional.
TRUNC|NOTRUNC	By default each field is truncated when moved from mainframe data format to CRLF delimited string format. (This operand has no effect on conversion to mainframe format.) Leading zeroes are removed from numeric data. If the field is zero, a single character "0" will be moved. Trailing spaces are removed from character data. If the field is all blanks, no data will be moved to the string. The default is TRUNC. Specify NOTRUNC to turn off field truncation.
SEP|NOSEP	Indicates whether the field is to be followed by the separator character as defined on the CONVERT command. SEP is the default. If you do not want the separator character added here, specify NOSEP. When converting from string to mainframe format, when a MOVE with NOSEP is encountered, the conversion process assumes that the NOSEP field is fixed length.

UPPER|**NOUPPER**	Use this operand to cause character data to be translated to upper case prior to conversion. This operand applies to conversions in both directions.
LOWER|**NOLOWER**	Use this operand to cause character data to be translated to lower case prior to conversion. This operand applies to conversions in both directions.
TRANSLATE|**NOTRANSLATE**	Indicates that translation is to occur on this specific field. This is intended for use in creating UNIX like binary files where character values need translation but binary numbers do not. If the TRANSLATE operand on the relevant CONVERT command is set to ALL ("TRANSLATE(ALL name)") then do not add the TRANSLATE option on the MOVE command as this will cause translation to occur twice and the resulting data will be garbled.
BINARY(BIG|LITTLE**)**	Optional parameter that indicates that the numeric value is to be handled as a binary value. Specify "BIG" or "LITTLE" for Big-endian (IBM mainframe) or Little-endian (Intel) format.
LENGTH(length**)**	You can optionally increase or decrease the field length of the resulting conversion by using this option. Normally the length will be the same as given for the corresponding field definition. This operand may be useful in conversions for UNIX-like binary files.
DECIMAL||NODECIMAL	Controls the insertion of a decimal character as needed in numeric data. The character used is taken from the CONVERT command. If the number of decimals for the field is zero ("PIC S9(4)" for example) no decimal point is inserted. The default is to add the decimal. Specify NODECIMAL to not insert the decimal character.
QUOTE(SINGLE|DOUBLE)	Use this to cause the data to be bracketed by either single or double quotation marks. If this operand is omitted, no quotes will be added.

EDIT(pattern**)**	This operand can be used to apply special rules to the field when converted – see the following for more information. The Edit pattern can be from 1 to 64 bytes in length.

The Edit pattern is a character string consisting of a single selector character ("@") and any number of optional characters as needed to correctly format the data. A couple examples should make this clear:

`@@@-@@-@@@@`	formats an American Social Security Number
`@@/@@/@@`	formats a date with slashes
`PRO:@@@@@@@@@`	adds the characters "PRO:" to the field.

Truncation rules from the TRUNC|NOTRTUNC operand do not apply when an Edit pattern is present in the MOVE command. Here leading zeroes and/or trailing spaces will be fed into the pattern as needed.

The Edit pattern can be applied to both numeric and character data.

Care must be taken to ensure that there are as many selectors, "@", as significant characters in the field or unpredictable results can occur. Leading zeroes for numeric data can often times be safely omitted, and depending on the field contents, trailing spaces may also be able to be safely ignored. It depends on the individual field formats and may require some experimentation on your part to develop a proper Edit pattern.

LITERAL

LITERAL ‘literal’ [SEP | NOSEP **]**

Literals can be inserted into the converted string as needed.

Literal	From 1ot 64 bytes of literal data can be entered into the command. The literal will be inserted as supplied into the converted string. If necessary hexadecimal data can be supplied by enclosing the data inside X’, for example LITERAL X’0B423132’ Be aware that this data will be translated according to the translate options given on the CONVERT command.
SEP|**NOSEP**	Indicates whether a separator character is to be added following the literal. The default is NOSEP.

When converting from string to mainframe data format, a LITERAL is assumed to be of the length specified in the command and ignored.

ADD

ADD string [ASCII | NOASCII]

The ADD command allows for the insertion of up to four special characters.

String	The string can be any combination of: **CR** - Carriage return, x'0D' **LF** - Line Feed, x'25' (or x'0A' in ASCII) **FF** - Forms Feed, x'0C'
ASCII|**NOASCII**	The Line Feed character (LF above) can take one of two values: x'25' in EBCDIC or x'0A' in ASCII. Use this operand to determine which is generated by HFS. ASCII is only needed if you are controlling translation at the MOVE level. If the CONVERT command specifies "TRANSLATE(ALL …" then you should let this operand default to NOASCII.

Typically, lines will be delimited by the presence of an ADD CRLF in the Conversion Rules.

If you omit the ADD CRLF command in the Conversion Rules, the conversion process will automatically insert one at the end of the record when converting from mainframe data to string data.

Likewise, when a CRLF is encountered when converting from string to mainframe data, end of record is assumed.

INSERT

INSERT BEFORE|AFTER FIRST|LAST|EVERY LINE|FIELD

The INSERT commands begins an INSERT group which allows for insertion of data at certain identifiable spots in the mainframe to string conversion process. **If the INSERT command is utilized, conversion from string to mainframe data will be impossible.**

The INSERT command provides for 12 different insertion points, not all of which are useable.

BEFORE FIRST LINE	Before any data from the mainframe is processed.
BEFORE FIRST FIELD	Before the first field identified by a MOVE command is processed during conversion. This is not necessarily the same spot as BEFORE FIRST LINE above.
BEFORE LAST LINE	**Not processed.**
BEFORE LAST FIELD	**Not processed.**
BEFORE EVERY LINE	Before every line (record) of mainframe data is processed.
BEFORE EVERY FIELD	Before every field but identified with a MOVE command.
AFTER FIRST LINE	After the first line (record) of mainframe data has been processed.
AFTER FIRST FIELD	After the very first field identified by a MOVE command is processed.
AFTER LAST LINE	After all data from the mainframe has been processed.
AFTER LAST FIELD	**Not processed.**
AFTER EVERY FIELD	After every field identified by a MOVE command is processed.
AFTER EVERY LINE	After every line (record) of mainframe data is processed.

The three conditions listed above as “Not Processed” cannot be identified in the File Conversion process.

All commands following the INSERT command are considered to be part of the insertion until an **ENDINSERT** command is encountered. For example, the INSERT commands below were used in generating HTML output :

```
INSERT BEFORE FIRST LINE
  LITERAL '<HTML><BODY BGCOLOR="CYAN">'       NOSEP
  LITERAL '<TABLE BORDER="1">'                NOSEP
 ENDINSERT
;
 INSERT BEFORE EVERY LINE
  LITERAL '<TR VALIGN=TOP>'  NOSEP
 ENDINSERT
  . . .
```

If needed, MOVE commands can also be placed within an INSERT group.

INSERT groups must be the first thing in the Conversion Rules following the CONVERT command.

ENDINSERT

The ENDINSERT command terminates an INSERT group – see example above.

END

END

When the END command is encountered, the preceding Conversion Rues are verified to ensure that all FIELD names are present in the associated File Definition, labels exist for GOTO and IF commands, and all TABLE commands are properly terminated.

You must clean up the verification errors, if any, before attempting to use the Conversion Rules. The actual conversion processor assumes that the Conversion Rules are correct and bad things will happen if an erroneous Conversion Rules are used.

IF

IF field condition {LITERAL(value)|FIELD(name)} THEN(label)

The IF command provides rudimentary logic capabilities in the Conversion Rules process. This can be useful for variable length records whose contents vary according to some record type field in the common part of the data. **If the IF command is present, conversion from string to mainframe data is not guaranteed to work properly.**

Field	This is the field name used in the associated File Definition for this IF test.
condition	Condition can be one of EQ, NE, LT, LE, GT or GE.
LITERAL(value)	If LITERAL is specified then you need to supply the literal value here. Watch your literal lengths here, if the length of the literal is not the same as the length of the corresponding field, the unequal condition will always be set. If necessary, you can define a special field in the File Definition to accommodate this length disparity. This operand is mutually exclusive with the FIELD operand below.
FIELD(name)	If FIELD is specified then you need to supply the field name here. As with literals above, watch your lengths here, if the lengthsof the two fields are not the same, the unequal condition will always be set. If necessary, you can define a special field in the File Definition to accommodate this length disparity. This operand is mutually exclusive with the LITERAL operand above.
THEN(label)	If the condition in the IF is satisfied you need to supply the label name of where processing is to continue within the Conversion Rules (see below). The label name can be from 1 to 16 characters in length.

GOTO

GOTO label

The GOTO command does what it implies. The next part of the Conversion Rules to be processed will be found at the label (see below) specified in the command. **If the GOTO command is present, conversion from string to mainframe data is not guaranteed to work properly.**

Label	This is the label name of where processing is to continue within the Conversion Rules (see below). The label name can be from 1 to 16 characters in length.

LABEL

LABEL name

The LABEL command provides a target in the Conversion Rules for either an IF or GOTO command.

Name	This is a unique name for the label. It can be from 1 to 16 characters in length.

Be careful with labels. Indiscriminate use of LABEL, GOTO and/or IF commands can result in infinite loops during conversion.

SKIP

The SKIP command indicates that the conversion process is complete at this point in the Conversion Rules. This may be useful with IF processing. **If the SKIP command is present, conversion from string to mainframe data is not guaranteed to work properly.**

SKIP

EXAMPLES

1. **Simple Conversion**

```
CONVERT 'FIXED TEST 1' 'FIXED TEST FILE 1'     -
         DECIMAL(PERIOD) SEP(',')
;
;           SIMPLE STRAIGHT-FORWARD COMMA SEPARATED STRING
;
 MOVE RCD-KEY                                    SEP NOTRUNC
 MOVE RCD-NAME                                   SEP
 MOVE RCD-TRANSACTION                            SEP
 MOVE RCD-SSN            EDIT('@@@-@@-@@@@') SEP
 MOVE RCD-TRAN-TYPE                              SEP
 MOVE RCD-TRAN-AMT                             NOSEP
 ADD  CRLF
 END
```

2. **Conversion using IF**

```
CONVERT 'FIXED TEST 4' 'FIXED TEST FILE 1'     -
         DECIMAL(PERIOD) SEP(',')
;
;           SELECTIVE EXTRACT OF FILE CONTENTS
;
 IF RCD-TRAN-AMT LT LITERAL('30.00') THEN(KEEPIT)
    SKIP
 LABEL KEEPIT
 MOVE RCD-NAME                                   SEP
 MOVE RCD-TRANSACTION   EDIT('@@-@@@@@@-@')  SEP
 MOVE RCD-TRAN-AMT                             NOSEP
 ADD  CRLF
 END
```

3. **Conversion with TABLE**

```
CONVERT 'VAR TEST 2' 'VARIABLE TEST FILE 1'     -
         DECIMAL(PERIOD) SEP(',')
;
;           TABLE file - one-to-one
;
 MOVE RCD-KEY       EDIT('@@@@-@@-@@@')      SEP NOTRUNC
 MOVE RCD-NAME                               SEP
 MOVE RCD-ENTRIES                            SEP
 TABLE BEG RCD-ACTION OCCURS(100)
 MOVE TBL-AMPM                             NOSEP
 MOVE TBL-TIME                               SEP NOTRUNC
 MOVE TBL-ACTION                             SEP
 TABLE END RCD-ACTION
 ADD  CRLF
 END
```

In this example there is a eon-to-one relationship between the mainframe data record and the string output. A string formatted by these rules can be converted back to the mainframe data record.

4. **Conversion with TABLE – individual lines**

```
CONVERT 'VAR TEST 1' 'VARIABLE TEST FILE 1'     -
         DECIMAL(PERIOD) SEP(',')
;
;           TABLE ENTRIES CONVERTED TO LINES
;
 TABLE BEG RCD-ACTION OCCURS(100)
 MOVE RCD-KEY         EDIT('@@@@-@@-@@@')    SEP NOTRUNC
 MOVE RCD-NAME                               SEP
 MOVE TBL-AMPM                             NOSEP
 MOVE TBL-TIME                               SEP NOTRUNC
 MOVE TBL-ACTION                           NOSEP
 ADD  CRLF
 TABLE END RCD-ACTION
 END
```

In this example, each individual entry in the table will generate an individual string in the output file. RCD-KEY and RCD-NAME reside in the fixed part of the record and are repeated on each output line. In this release of HFS, conversion of string to mainframe data will generate individual records as well and you will need to write your own code to merge the records together if necessary.

Loading SD Files

CSIHFLOD can be used to load a variety of sequential files into an HFS, or unload from an HFS.

CSIHFLOD is <u>not</u> provided as part of *HFS-Lite*.

Use the following JCL as a guide for running this program.

```
* $$ JOB JNM=HFSSD,CLASS=Z
* $$ LST CLASS=L
// JOB HFSSD
/* OPTION NODUMP
// DLBL CSIHFDT, …          if needed
// EXTENT SYS019, …         if needed
// LIBDEF *,SEARCH=( …      as required
// DLBL CSIHBAK, …          SD Source file
// EXTENT …                 as required
// EXEC CSIHFLOD,SIZE=AUTO
 ACCESS CSIHFDT
 SD CSIHBAK RECFM(VARBLK BLKSIZE(1024) ) -
    TO ('/LOADED FILE' REPLACE)
/*
/&
* $$ EOJ
```

Some commands also present in CSIHFBAT were duplicated here as well to simplify processing.

CSIHFLOD – Commands

OPTION

OPTION [ALLOWNAME | <u>NOALLOWNAME]</u>

ALLOWNAME | **NOALLOWNAME**	If **ALLOWNAME** is specified, special characters such as the colon are allowed to be present in the file and/or directory names. The default, **NOALLOWNAME**, edits the names as described in the HFS manual.

ACCESS

This is required for anything other than INITIALIZE and tells the batch program the DLBL name of the file to be used

ACCESS filename [**SEP(‘**char**’)]**

SEP(‘char’)	**SEP** can be used to change the directory separator character from the default, forward slash (‘/’) to any character you want except for period (‘.’) which is reserved for HFS internal use as a separator for the file name extension.

MAKEDIR

MAKEDIR ‘directory name’

Makes a directory. The name can be up to 256 bytes long. If it contains embedded spaces, slash or parenthesis enclose it in single quotes (for example: ‘TEST DIRECTORY ONE’). Otherwise, no quotes are needed.

Command can be abbreviated as MD.

CHANGEDIR

CHANGEDIR ‘directory name’

Changes to a different directory. The name can be up to 256 bytes long. If it contains embedded spaces, slash or parenthesis, enclose it in single quotes (for example: ‘TEST DIRECTORY ONE’). Otherwise, no quotes are needed.

Command can be abbreviated as CD,

SD

The SD command can be used to load a Sequential Disk file to an HFS or extract a Sequential Disk file from an HFS.

SD name RECFM(type LRECL(num) BLKSIZE(num)) –
{TO|FROM} (…)

Name	This is the seven character DLBL name. A Corresponding DLBL and EXTENT must be present in the JCL or available in standard labels.
RECFM(type	Describes the type of sequential file to be accessed. It can be one of VARBLK – Variable, Blocked VARUNB – Variable, Unblocked FIXBLK – Fixed, Blocked FIXUNB – Fixed, Unblocked SPNBLK – Spanned, Blocked SPNUNB – Spanned, Unblocked UNDEF – Undefined Note for Undefined, unpredictable results may occur – see "Appendix B – Undefined Files" for more information.
LRECL(num)	For fixed length records you must supply the logical record length. This parameter is ignored for other types.
BLKSIZE(num)	You must always supply the block size of the Sequential Disk file. BLKSIZE cannot exceed 32767.
TO|FROM(…)	TO tells CSIHFLOD to write an SD file to the HFS. FROM tells CSIHFLOD to write an HFS file to the SD file. This parameter is used in several commands and described separately below.

LIBR

This command allows you to load a member from LIBR into an HFS. Only text members may be loaded into an HFS.

LIBR lib.sublib name.ext TO(…)

Lib.sublib	This is the VSE library and sublibrary that contain the member to be loaded.
Name.ext	This is name and extension of the VSE library member to be loaded. This name can be generic. For instance "*.L" will load all members From the VSE library.sublibrary with extension "L." File names will be the same as found on the VSE library, the name on the TO segment will be ignored (you will still have to supply a non-blank name to make the parser happy, but it will be ignored).
TO(…)	This parameter is used in several commands and described separately below. Currently you can only move a member from a VSE library to an HFS file.

TO|FROM (…)

The TO|FROM segment is used in both the SD and LIBR commands. Its requirements are the same for both commands.

TO|FROM (‘name’ [ASCII | NOASCII} –
[REPLACE **|** NOREPLACE] –
[SAFE | NOSAFE**]** APPEND | NOAPPEND])

Name	Specify here the name of the file on the HFS. IF the name begins with a separator character, the name is assume to be the fully qualified path name for the file. Without the leading separator, the file will be placed in the current directory.
ASCII|**NOASCII**	You can request ASCII translation for the file as needed. The default is NOASCII which is no translation. Option is ignored on a FROM segment.
REPLACE|NOREPLACE	Determines what to do when a duplicate file is encountered. Specify REPLACE to delete the existing file and replace it with the new contents. Specify NOREPLACE to stop processing of the load operation. Option is ignored for a FROM segment.
SAFE|**NOSAFE**	If REPLACE is specified above, you can qualify the replacement with this option. SAFE will first rename the existing file before attempting the load. If any problems occur during the load, the original file is restored. Option is ignored for a FROM segment.
APPEND|**NOAPPEND**	Determines whether the file is to be appended to an existing file in the HFS. Option is ignored for a FROM segment.

Examples

SD CSIFILE RECFM(VARBLK BLKSIZE(20000)) –
TO(‘/directory 1/csifile.txt’ REPLACE SAFE)

This will load the contents of CSIFILE into “/directory 1/csifile.txt” using safe replacement of supplicates. The file is a variable blocked file.

CD ‘/Sample Directory’
SD CSIFIL1 RECFM(FIXBLK LRECL(100) BLKSIZE(1000)) –
TO(‘appended files’ REPLACE)
SD CSIFIL2 RECFM(FIXUNB LRECL(100) BLKSIZE(100)) –
TO(‘appended files’ APPEND)
SD CSIFIL3 RECFM(FIXBLK LRECL(100) BLOSIZE(4000)) –
TO(‘appended files’ APPEND)

These commands load the contents of three files, CSIFIL1, CSIFIL2 and CSIFIL3 into “/Sample Directory/appended files”. Each file is fixed length but has different block sizes. The final contents of the HFS file will be the contents of the three files specified in the order shown. This file can be processed by any program with any block size through the use of the HFS File Interception Facility.

LIBR CSILIB.T HOOBI.A TO(‘/hoobie file’)

VSE library member HOOBI.A from Library CSILIB.T will be written to the HFS as “hoobie.file.” File is assumed to be 80 byte fixed length records.

SD CSIFILE RECFM(VARBLK BLKSIZE(20000)) –
FROM(‘/directory 1/csifile.txt’)

Here we will unload the file written above and move it from the HFS to a standard Sequential Disk file.

HFS File Recovery

Due to its size and complexity, the Recovery command is discussed separate from the other commands honored by CSIHFBAT. The RECOVER command runs under CSIHFBAT discussed above in the section titled "HFS File Management."

RECOVER mode [AUTO | NOAUTO] [MSG | NOMSG] - [RESET | NORESET] [SYS(LST)]

.

<table>
<tr><td>Mode</td><td>There are three modes for recovery:
<table>
<tr><td>QUICK</td><td>QUICK recovery ensures that the HFS directory is intact and little else. Only the directory itself is checked for errors.</td></tr>
<tr><td>STANDARD</td><td>STANDARD does everything QUICK does plus it verifies that at least the first record of every file exists and is correct.</td></tr>
<tr><td>DEEP</td><td>DEEP verifies the directory and attempts to read each file contained within the HFS in its entirety. In addition, DEEP recovery will remove any stray records (records identified in the FAT but not present on any of the active chains in the HFS). This is the most through recovery. It is also the most time consuming. No other tasks/partitions can be accessing the file while DEEP recovery is running.</td></tr>
</table></td></tr>
<tr><td>AUTO | NOAUTO</td><td>This setting applies to QUICK or STANDARD recovery. The option controls the processing whenever a serious error is encountered. If AUTO is specified, the recovery process will immediately initiate a DEEP recovery of the HFS file. If NOAUTO is specified, processing depends on the MSG option below.</td></tr>
<tr><td>MSG | NOMSG</td><td>This setting applies to QUICK or STANDARD recovery. If AUTO is specified (above) this operand has no meaning. This option determines whether or not the console request will be issued when serious errors occur. NOMSG means no console request.</td></tr>
<tr><td>RESET | NORESET</td><td>Use the RESET option to restore operations to the HFS file following a failure in a previous RECOVER process. If RESET is specified, all other options on the RECOVER command will be ignored.</td></tr>
</table>

SYS(LST**)**	This setting determines where the Recovery Control Report is to be written. Several options are possible: LST — This is the default and indicates SYSLST 000 — 000 is magic and indicates no report. This is not recommended. 001-241 — The corresponding SYSnumber is assumed to be assigned to a SYSLST device. LOG — Put the control report on the console - not recommended.

You shouldn't get too big a warm-and-fuzzy out of successfully passing **QUICK** recovery – it doesn't do much. Even **STANDARD** is somewhat less than desirable. **DEEP** recovery is the only mode that ensures complete recovery of the HFS file.

And then reality intrudes. **DEEP** recovery may take an intolerably long time for larger files should the recover process be put on a periodic schedule. Especially since the HFS file will be intact most of the time.

The AUTO|NOAUTO option helps in this regard. By setting this to AUTO, you can run the recovery in one of its lighter modes and let it automatically switch to the most thorough processing should an error be detected. For example **RECOVER STANDARD AUTO** will run reasonably quickly and ensure that you can at safely access the directory and get to file identifications for everything currently contained in the HFS. Even so, it is probably wise to occasionally run a DEEP recovery to make certain that the file is clean.

The Recovery process produces a control report listing the actions it has taken on your behalf. A sample of the report follows.

```
02/25/04 11:56:57   RECOVERY FOR FILE CSIHFDT                          PAGE      1
BEGIN RECOVERY - D E E P
CSIHFRCV-10 ONE OR MORE FILE RECORDS MISSING - FILE DELETED
       FILE  F='DIRECTORY THREE/FILE SEVEN'
FILES DELETED
            F='DIRECTORY THREE/FILE SEVEN'
         1 FILE DELETES     PROCESSED
DIRECTORY ENTRIES REMOVED
            F='DIRECTORY THREE/FILE SEVEN'
         1 DIRECTORY REMOVALPROCESSED
RECOVERY COMPLETE
         8  DIRECTORIES
         8  FILES
        51  STRAY RECORDS DELETED
        52  CORRECTIONS MADE
```

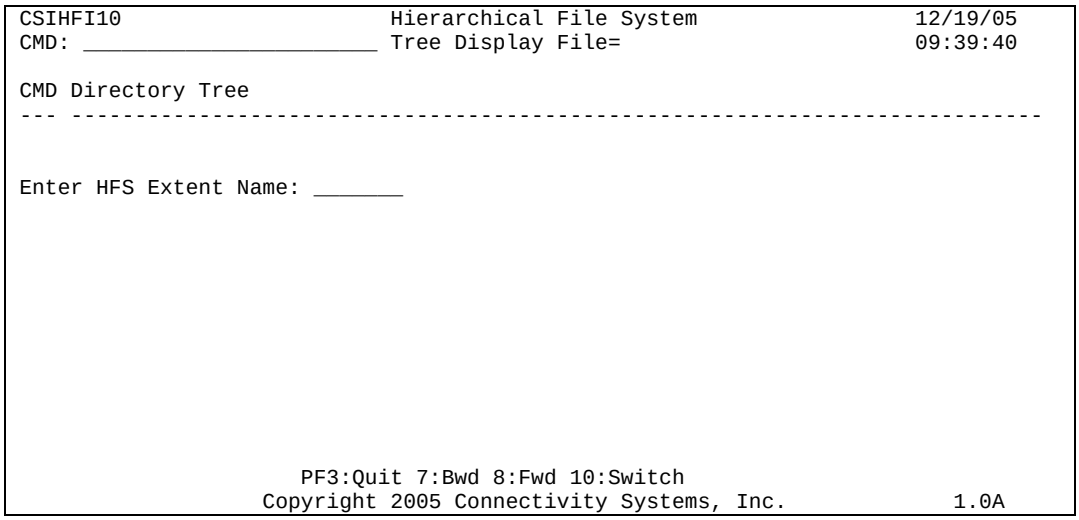
HFS Online

A single CICS transaction is provided as part of HFS installation. This transaction can be used to access any HFS in your system provided, of course, that the relevant DLBLs and EXTENTs are known to CICS, wither in the CICS JCL or standard labels.

HFS Online is <u>not</u> provided with *HFS-Lite.*

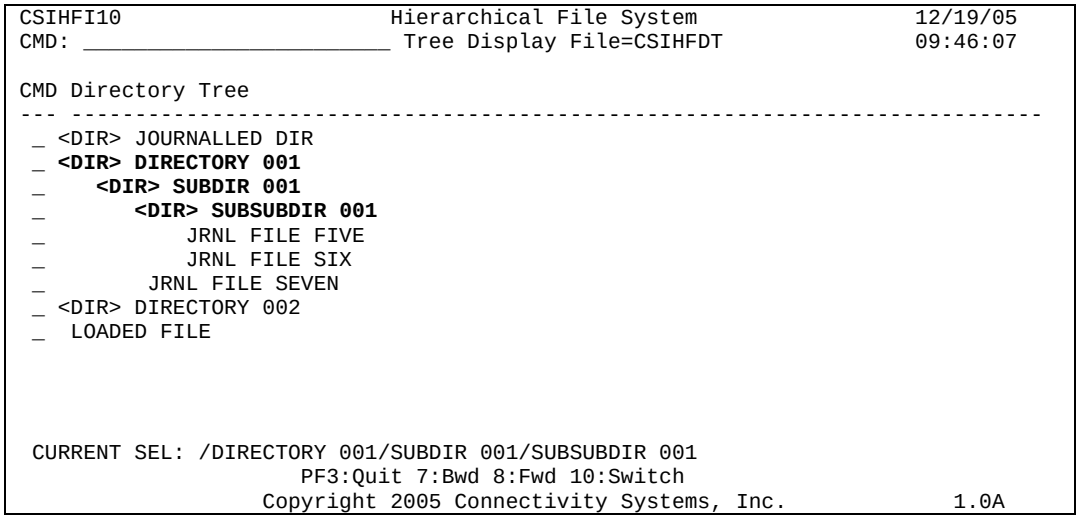
The transaction is **HFIL**. You are free to change this, as noted in the installation procedure, so contact your system administrator for the proper transaction to use.

Initial Screen



In order to proceed you must enter the seven character file name for the HFS. You can enter that name in the CMD area in the upper left hand corner or in the body of the screen, whichever you prefer. Press enter and you will be presented with the "Tree View."

Tree View



The Tree View shows the directory and file structure of the HFS. The HFS being viewed is identified on the second line of the display. Each level of the directory structure is indicated by indentation as shown in the sample screen above. Directory entries are prefixed by "<DIR>."

At the bottom of the screen the current path is shown. The path is also highlighted in the Tree View itself.

On the second line of the screen, several commands are recognized:

ACC name	Use the ACC command to switch to a different HFS extent. The name that follows is the seven character DLBL name of the HFS extent. There must be a space between "ACC" and the name.
BWD	Scroll backwards just like PF7.
FWD	Scroll forwards just like PF8.
QUIT	Quit the HFIL transaction. This is also performed by the CLEAR and PF3 keys.
REFRESH	Refreshes the current view at the ROOT directory level. Any currently expanded subdirectories will be collapsed as part of this command's processing. Use the "R" command, below, to refresh individual subdirectories.

On each line, the following commands are recognized:

+	A plus sign will a directory to be expanded (as shown above).
-	A minus sign will collapse a previously expanded directory and all subdirectories with it.
V	A 'V" for **V**iew can be entered to display the contents of the individual HFS file. (A sample View is shown below.)
D	Deletes the current file from the HFS.
R	Causes the contents of the selected directory to be refreshed on the screen. Anything outside of this subdirectory is unaffected by this command.
/	A slash ('/') will position the display with the line containing the slash as the first shown on the screen.

If you wish to perform maintenance to the HFS online, such as make directory, you will need to toggle the display to the Command Line (see below).

Available PF keys are:

PF3	Exit the Tree View display. You can also press the **CLEAR** key to exit this display.
PF7	Scroll backward through the Tree View.
PF8	Scroll forward through the Tree View.
PF10	Toggle the transaction to the Command Line presentation (see below).

View Display

```
CSIHFI30                      Hierarchical File System               12/19/05
CMD: _____________________ File=/VARIABLE CONVERT FILE               09:59:29
-------------------------------------------------------------------------------
    00 30303031 2D30302D 3135382C 47494C4C |0001-00-158,GILL|
    10 4947414E 2C413038 34352C42 524F4B45 |IGAN,A0845,BROKE|
    20 2050524F 46455353 4F525320 4641564F | PROFESSORS FAVO|
    30 52495445 20434F43 4F4E5554 0D0A3030 |RITE COCONUT..00|
    40 30312D30 302D3135 382C4749 4C4C4947 |01-00-158,GILLIG|
    50 414E2C41 30383436 2C52414E 20415741 |AN,A0846,RAN AWA|
    60 5920414E 44204849 440D0A30 3030312D |Y AND HID..0001-|
    70 30302D31 35382C47 494C4C49 47414E2C |00-158,GILLIGAN,|
    80 41313030 302C4449 5343564F 56455245 |A1000,DISCVOVERE|
    90 44205452 49424520 4F462043 414E4E49 |D TRIBE OF CANNI|
    A0 42414C53 204F4E20 49534C41 4E440D0A |BALS ON ISLAND..|
    B0 30303031 2D30302D 3135382C 47494C4C |0001-00-158,GILL|
    C0 4947414E 2C413130 30312C52 414E2041 |IGAN,A1001,RAN A|
    D0 57415920 414E4420 4849440D 0A303030 |WAY AND HID..000|
    E0 312D3030 2D313538 2C47494C 4C494741 |1-00-158,GILLIGA|
    F0 4E2C5030 3432322C 53415645 44205448 |N,P0422,SAVED TH|
   100 45204441 59205448 524F5547 4820534F |E DAY THROUGH SO|
   110 4D452053 494C4C49 4E455353 0D0A3030 |ME SILLINESS..00|
   120 30312D30 302D3234 352C5448 4520534B |01-00-245,THE SK|
   130 49505045 522C4130 3833322C 5741434B |IPPER,A0832,WACK|
   140 45442047 494C4C49 47414E20 4F4E2048 |ED GILLIGAN ON H|
   150 45414420 57495448 20484154 0D0A3030 |EAD WITH HAT..00|
   160 30312D30 302D3234 352C5448 4520534B |01-00-245,THE SK|
   170 49505045 522C4130 3833332C 5741434B |IPPER,A0833,WACK|
   180 45442047 494C4C49 47414E20 4F4E2048 |ED GILLIGAN ON H|
   190 45414420 57495448 20484154 0D0A3030 |EAD WITH HAT..00|
   1A0 30312D30 302D3234 352C5448 4520534B |01-00-245,THE SK|
   1B0 49505045 522C4130 3833342C 5741434B |IPPER,A0834,WACK|
   1C0 45442047 494C4C49 47414E20 4F4E2048 |ED GILLIGAN ON H|
   1D0 45414420 57495448 20484154 0D0A3030 |EAD WITH HAT..00|
   1E0 30312D30 302D3234 352C5448 4520534B |01-00-245,THE SK|
   1F0 49505045 522C4130 3834382C 44495343 |IPPER,A0848,DISC|
   200 4F564552 45442042 524F4B45 4E20434F |OVERED BROKEN CO|
   210 434F4E55 540D0A30 3030312D 30302D32 |CONUT..0001-00-2|
   220 34352C54 48452053 4B495050 45522C41 |45,THE SKIPPER,A|
   230 30383439 2C4C4F4F 4B494E47 20464F52 |0849,LOOKING FOR|
   240 2047494C 4C494741 4E20534F 20484520 | GILLIGAN SO HE |
   250 43414E20 5741434B 2048494D 0D0A3030 |CAN WACK HIM..00|
  ASCII                    PF3:Quit 7:Bwd 8:Fwd
                  Copyright 2005 Connectivity Systems, Inc.            1.0A
```

The HFS file is displayed in hex and character as shown above.

The fully qualified path name of the file being displayed is shown at the top of the screen. IF the file name will not fit the available space, it will be shorted on the left and replaced with an ellipses (“…”).

There are several commands available in the CMD area at the top of the screen.

ASCII	Causes the display portion of each line to be translated from ASCII. This translation status is shown at the bottom left of the screen. (Note: the sample screen display above has had the ASCII command applied.)
EBCDIC	Causes the display portion of each line to be treated as if it were in EBCDIC. This is the default state and the initil display of any file will assume EBCDIC presentation.
TOP	Positions the display at the first byte of the file.
BOT	Positions the display so that the last bytes of the file are displayed.
+nnn	Positions the display forward by the number given. The number is assumed to be in hexadecimal.
-nnn	Positions the display backward by the number given. The number is assumed to be in hexadecimal.
POSnnn	Positions the display to the offset supplied with the command. The offset is assumed to be in hexadecimal.

Three PF keys are available.

PF3	This will return you to the Tree View Display. You can also press the **CLEAR** key to return to the Tree View.
PF7	Scroll backward in the file.
PF8	Scroll forward in the file.

Command Line

```
CSIHFI10                      Hierarchical File System                  12/19/05
CMD: _____________________ CMD Line     File=CSIHFDT                  10:50:33
 05/11/18 06:42:28  <DIR> DIRECTORY 001
 05/11/18 06:42:28  <DIR> DIRECTORY 002
 05/11/18 06:42:28  25.0K LOADED FILE
 05/12/19 10:41:36  <DIR> HI THERE
 MD NEW Directory
 O.K.
 DIR
 Directory of: ROOT
 05/11/18 06:42:28  <DIR> JOURNALLED DIR
 05/11/18 06:42:28  <DIR> DIRECTORY 001
 05/11/18 06:42:28  <DIR> DIRECTORY 002
 05/11/18 06:42:28  25.0K LOADED FILE
 05/12/19 10:41:36  <DIR> HI THERE
 05/12/19 10:49:28  <DIR> NEW DIRECTORY
 STATS
            FILE: C S I H F D T
            DATE: 1 1 / 1 8 / 0 5
            TIME: 0 6 : 4 2 : 2 7
     RECORDS MAX.:          7,800
     RECORDS USED:             35      0 0 . 4 4 %
     CACHE   HITS:              0
     CACHE MISSES:              0
      FILE  READS:             98
      FILE WRITES:              3
 cd directory 001
 O.K.
--------------------------------------------------------------------------------
 RC=00   CMD:
 CURRENT DIR: /DIRECTORY 001
                           PF3:Quit 10:Switch
                    Copyright 2005 Connectivity Systems, Inc.            1.0A
```

This screen was developed to test HFS in a CICS environment and as an exercise in silliness to emulate PC command line processing. Due to the restrictions of 3270 environment it is of limited usefulness.

As with PC DOS command line, the most recent response is shown at the bottom of the display area. As commands are entered, least recent activity will scroll of the screen at the top.

In the sample screen above, the following commands were issued:

DIR - directory of ROOT has partially scrolled off the screen.
MD – created a new subdirectory “NEW DIRECTORY”
DIR – ensuring that new subdirectory was added
STATS – obtained file statistics for the HFS
CD – positioned the current directory to “directory 001”

Any commands you choose to use should be entered at the CMD area near the bottom of the screen. The documented commands are:

DIR	Responds with a directory listing as illustrated in the sample screen above.
CD name	Change the current directory to the directory named in the command.
MD name	Make a new directory of the name in the command.
RD name	Remove a directory of this name. The directory must be empty prior to removal.
STATS	Display the HFS file statistics as shown in the sample screen.

There simply isn't enough room on the 3270 display to implement other HFS commands such as rename.

CSI recommends that you use the HFS batch utility program for HFS maintenance.

Two PF keys are recognized:

PF3	Exit the Command Line display. You can also press the **CLEAR** key to exit this display.
PF10	Toggle the transaction to the Tree View presentation (see above).

HFS API

HFS provides an **A**pplication **P**rogramming **I**nterface for your use. The API works in both CICS and batch for either assembler or COBOL programs.

The HFS API is <u>not</u> available with *HFS-Lite*.

Four copy books were installed for the API:

HFSAPI.A - API parameter list (assembler)
HFSAPI.C - API Parameter list (COBOL)
HFSAPID.A – API Directory Response (assembler)
HFSAPID.C – API Directory Response (COBOL)

The fields and use of these copy books is described in greater detail below.

Before using the API be sure to read the "HFS API Gotchas" section below.

Accessing HFS Through the API

VSE BATCH

COBOL : CALL 'CSIHFAPB' USING the.parameter.list

```
ASM:        LA    R1,the.parameter list
            ST    R1,some.field
            LA    R1,some.field
            LA    R13,save.area    DEFINED AS DS 9D
            L     R15,=V(CSIHFAPB)
            BASR  R14,R15
```

VSE CICS

```
EXEC  CICS LINK PROGRAM(CSIHFAPI)          -
       COMMAREA(the.parameter.list)        -
       LENGTH(length.including.data.buffer)
```

NOTE: you must use **CSIHFAPI** in CICS and **CSIHFAPB** in batch.

(See "Installation, Step 5 – CICS Table Additions" for the necessary CICS table entries to enable the API.)

HFS API Parameter List

Communications with the HFS API is done using the parameter list as found in the4 HFSAPI.A and/or HFSAPI.C copy books. The fields contained in the parameter list are described below – both COBOL names are shown first with the Assembler names on the second line.

Field Name	Format	Description
HFSAPI-ID CAHFSID	PIC X(8)	This must be set to "**CSIHFAPI.**"
HFSAPI-FUNC CAHFUN	PIC X	The function code is required. It is described in more detail below.
HFSAPI-RETURN CAHFRET	PIC X	The completion status of your request is returned here. It can be: 0 – O.K. 4 – End Of File 8 – Error
HFSAPI-OPTBYTE1 CAHFOP1	PIC X	Some functions allow for optional parameters. See the function description for more information.
HFSAPI-OPTBYTE2 CAHFOP2	PIC X	Some functions allow for optional parameters. See the function description for more information.
HFSAPI-SEP CAHFSEP	PIC X	The default directory separator is forward slash ('/'). You can optionally change it by putting your preferred separator here. NOTE: This is only recognized on the "Open HFS Access" request.
HFSAPI-HFS-NAME CAHFNM	PIC X(7)	This is the HFS name you want to access. A corresponding DLBL and extent must be available in the partition.
HFSAPI-API-TOKEN CAHFTOK	PIC X(4)	This is the HFS API token which is returned to you by the *Open HFS Access* request. You need to supply it on all other calls. (See Token Handling, below for more information.
HFSAPI-BUF-LENGTH CAHFBLN	PIC S9(9) COMP	Put the length of your buffer area here. (See the individual function descriptions below for instructions on how to use this field.)
HFSAPI-RET-LENGTH CAHFRLN	PIC S9(9) COMP	The length of data returned to you by the API is placed here. This may not be the same size as the buffer length above. It can even be zero. (See *Get File* request, for more information.)

HFSAPI-ERRMSG1 CAHEM1	PIC X(80)	If an error is detected in the API, a message will be placed here. The first three bytes of the message are numeric and can be interpreted programmatically.
HFSAPI-ERRMSG2 CAHEM2	PIC X(80)	Occasionally, a 2nd message will also be present.
HFSAPI-ERRMSG3 CAHEM3	PIC X(80)	On rare occasions, a 3rd line will be generated by the API.
HFSAPI-FMT CARECFM	PIC X	You can specify either "F" for fixed or "V" for variable here. (See the *Put File* request, for more information.)
HFSAPI-FIX-LEN CALRECL	PIC S9(4) COMP	If the format (above) is "F" place the record length here.
HFSAPI-NAME1 CAHFNM1	PIC X(256)	Use this field to supply the file name for the API function you are using. (Assembler programs should pad this out with either spaces or binary zeroes.)
HFSAPI-NAME2 CAHFNM2	PIC X(256)	For the *Rename File* request the new file name is placed here. This field is ignored for all other API requests.
HFSAPI-BUFFER CAHFBUF	User-defined	Your data buffer starts here. The format and contents of this area are up to you. NOTE: the directory requests, "K" and "L", are returned in this area formatted as described below.

HFS API Function Requests

A – Open Access to HFS

This must be the first request you make of the API in any HFS session. This command starts an HFS session.

HFSAPI-HFS-NAME (CAHFNM) is required.

HFSAPI-SEP (CAHFSEP) is optional and if supplied changes the separator character to that which you provide here.

Upon successful completion, the HFS API returns **HFSAPI-API-TOKEN** (CAHFTOK). This value must be supplied to all subsequent requests for this HFS API session. (See Token Handling below for more information.)

B – Open File

This request is not necessary and is supplied only because too many programmers have difficulty wrapping their brains around the concept of being able to read or write a file without a file open. You do not need to issue this request in order to access files that are contained within the HFS.

HFSAPI-NAME1 (CAHFNM1) is required. However, it may be overridden by a subsequent *Get File*, *Put File* or *End File* request.

HFSAPI-FMT (CARECFM) and **HFSAPI-FIX-LEN** (CALRECL) are optional. They are meaningless for a subsequent Get File request, and may be overridden by the *End File* request when writing to the HFS.

C – Get File

Use this request to read information stored in the HFS.

HFSAPI-NAME1 (CAHFNM1) is required.

HFSAPI-BUFFER (CAHFBUF) will contain the results of the Get File request to a maximum number of bytes as specified by **HFSAPI-BUF-LENGTH** (CAHFBLN). The API cannot detect a storage overlay, it is up to you to ensure that there is sufficient space in the buffer to hold the number of bytes you specify in the length.

Upon successful return, the API will place the number of bytes it placed in the buffer in **HFSAPI-RET-LENGTH** (CAHFRLN). Since it is unlikely that the file will contain an integral multiple of data buffers, this number could be less than what you supplied in **HFSAPI_BUF_LENGTH** (CAHFBLN) above.

Normally, data is returned along with the End Of File indication, so you need to check the return length to see if any data was supplied when an End Of File is encountered. It is possible that End Of File may return zero bytes, but normally there is some data returned.

D – Put File

Use this request to write data to the HFS.

HFSAPI-NAME1 (CAHFNM1) is required.

HFSAPI-BUFFER (CAHFBUF) is assumed to contain the data to be written to the HFS for a length of **HFSAPI-BUF-LENGTH** (CAHFBLN).

HFSAPI-OPTBYTE1 (CAHFOP1) can be used to control processing when a duplicate file is encountered in the HFS. It can be one of:

A – Append this data to the existing file
D – Replace the existing file with new data
R – Rename the file (see below).

Any other value placed in this filed will receive an error with message "210 – Duplicate File" being placed in **HFSAPI-ERRMSG1** (CAHEM1).

If **R**ename is specified, the file name will be examined from right to left looking for the character string "000". When found, this value will be incremented until arriving at a unique file name. The results of the rename processing will be reflected to you in **HFSAPI-NAME1** (CAHFNM1) whose contents will be changed to the actual name used by the Put File request.

It is not necessary to have a buffer large enough to transmit the entire file in one *Put File* request. Each subsequent *Put File* request, until terminated by an *End File* request, will add data to the current file in the same manner as VSE LIOCS. You can issue as many *Put File* requests as needed to the same file.

E – End File

It is necessary to terminate both the *Put File* and *Get File* requests with an *End File* request. This frees up storage and removes internal locks so other tasks may process the file you have just read or written.

HFSAPI-NAME1 (CAHFNM1) is required.

HFSAPI-FMT (CARECFM) and **HFSAPI-FIX-LEN** (CALRECL) are optional. They are meaningless when ending a *Get File* request.

F – Delete File

Use this request to remove a file from the HFS.

HFSAPI-NAME1 (CAHFNM1) is required and names the file to be deleted.

Standard HFS locking rules apply. If the file is currently being processed by your task or some other task in the VSE machine, the delete will fail.

The *Delete File* operation is irreversible. There is no undelete capability in HFS.

G – Rename File

Use this request to rename a file in the HFS.

HFSAPI-NAME1 (CAHFNM1) is required and is the current file name.

HFSAPI-NAME2 (CAHFNM2) is required and is the new file name.

You can use the *Rename File* function to move a file around in the directory structure. If you do decide to change the location of the file, then **HFSPI-NAME2** (CAHFNM2) must be the fully qualified path name beginning with a separator.

H – Change Directory

Use this request to change to a different directory in the HFS file structure.

HFSAPI-NAME1 (CAHFNM1) is required.

If the name begins with a separator, it is assumed to be the fully qualified path name starting from the HFS root directory. Otherwise, HFS assumes that the name refers to a subdirectory of the current directory.

To change to the root directory, use a one byte name consisting of the separator character only. You can back up one directory by providing the special name ".." (two periods) much like you can on a PC.

I – Get Current Directory

Use this request to obtain the current directory.

The current directory is returned as the fully qualified path name beginning with a leading separator in **HFSAPI-BUFFER** (CAHFBUF) which is assumed to be at least 256 bytes long.

J – Make Directory

Use this request to make a new directory in the HFS file structure.

HFSAPI-NAME1 (CAHFNM1) is required.

If the name begins with a separator, it is assumed to be the fully qualified path name starting from the HFS root directory. Otherwise, HFS assumes that the name refers to a subdirectory of the current directory.

The current directory is not changed as a result of this request.

K – Get Directory (first)

Use this request to start to retrieve the contents of the current directory.

The Directory process is split into two commands, this one starts the process and *Get Directory (next)* continues it. See the following command for more information.

L – Get Directory (next)

Use this request to continue to retrieve the contents of the current directory.

In response to this request, the API returns a single directory entry in **HFSAPI-BUFFER** (CAHFBUF) which is assumed to be at least 316 (x'13C') bytes long. This is a formatted response which is described in detail below.

When End Of File is raised, a directory response is not returned. This is different from the way End Of File is treated by the Read File command.

No special effort needs to be made to terminate directory processing. When you have gotten what you need, simply make another request of the API. Once you do so, however, the directory process cannot be resumed without starting over with the *Get Directory (first)* command. As a result, if you will need to retrieve multiple directory entries, such as a list of files, you should examine the directory first and same the relevant file names in a table someplace before attempting to access any individual file.

M – Backout File

Use this request to clean up the HFS following an uncompleted write operation.

If for some reason you decide that you will not want to write a file, using the *Put File* request, after all, you should issue this request to clean up the HFS. This is primarily intended for CICS HANDLE ABEND and batch STXIT routines.

Failure to backout an incomplete Put Request will lead to the accumulation of stray records in the HFS. These can be removed using the batch RECOVER command, but it is preferable to handle it within the API if the situation can be recognized.

Y – Options

Use this request to alter the default operation of HFS for this session.

Currently only two options are supported. More may be added in the future.

The options are to be placed in **HFSAPI-BUFFER** (CAHFBUF) which is assumed to be at least 32 bytes long. This area should be padded with spaces to the 32 byte limit. Options consist of one or two character strings, separated by one or more spaces. The possible strings are:

JOURNAL **NOJOURNAL**	Specify "JOURNAL" to turn on HFS Journaling for write activity, Specify "NOJOURNAL" to turn off journaling. The HFS default is "NOJOURNAL."
ALLOWNAME **NOALLOWNAME**	Use "ALLOWNAME" to allow special characters in the file and/or directory names. The HFS default is "NOALLOWNAME."

You can set and reset these options as often as needed within a single HFS session.

Z – Close Access to HFS

Use this request to terminate the HFS session.

This request frees up resources associated with the HFS session. This is primarily a memory usage problem and need not be done in batch if your program is about to quit anyway. It is much more critical in CICS.

HFS acquires approximately 11K of SHARED storage in CICS. 10K can go anywhere, above or below the line. 1K must be below the line. During the midst of a get or put process, an additional 4k of go anywhere storage is acquired. SHARED storage does not go away with the end of a CICS task but must be explicitly freed by the program that acquired it.

If you do not issue the *Close Access to HFS* request, your CICS DSA will slowly fill up with these little bits of SHARED storage. Therefore, in CICS, you really should code a HANDLE ABEND routine to close off the HFS API in the event of a program failure.

HFS API Token Handling

When you start an HFS session using the *Open HFS Access* request, a four byte token is returned to you by the API. You must supply that same token with all subsequent requests to the API.

This is not a big problem in a batch environment, but in CICS you may have to pass the token around in the COMMAREA between programs and/or pseudo-conversational tasks. Provided CICS remains up, the token will survive for years of inactivity.

Once you have a token, you can use it for any of the other API requests. It is not necessary and strongly discouraged to obtain a new token for each HFS activity you wish to perform. With your original token you can read as many files as you want, write as many files as you want, and browse as many directories as you want, rename files, make directories, delete files, provided that you do these things one at a time. A file write (or read) can even take several pseudo-conversational task to complete.

Under the covers, HFS is a state engine. For example, the first *Put File* request sets the HFS internal state to Put. All subsequent put requests will continue to write data to the same file until an *End File* request is encountered. After the *End File* request is processed, HFS is in no particular state and ready to be redirected to some other activity.

This processing can continue indefinitely on the same token. The only reason to obtain a second token is if it is necessary to read (or write) two or more files simultaneously. In which case, you will need a unique token for each file that is being processed simultaneously. But if all you need to do is read one or more files sequentially, then a single token can handle it: read one file to End of File, issue the *End File* request and then start reading the next file.

You absolutely must use the same token to complete a Directory, Put or Get operation that you used to start it with. If you do not, you will simply get the same record (directory entry) over and over, or lose all but the last data written to the HFS.

File Formats (HFSAPI-FMT)

This format indicator may or may not be present depending on how the file was created. If you want to set this yourself, be careful. Fixed format is straight forward. Variable, on the other hand, is not.

Variable format indicates that the file is formatted like a normal VSE variable unblocked file with a LLBB preceding the actual record. The LL part contains the length of the variable record plus the 4 byte length of the LLBB itself. Spanned processing is not supported in HFS so the BB part must always be x'0000'.

If you want to set format to Variable, then you must ensure that its contents match the description above.

HFS API Directory Response

The directory requests, “K” & “L”, return a formatted response in the API parameter buffer. This response is defined by copy books HFSAPID.A and HFSAPID.C for assembler and COBOL respectively. The individual fields of the response are:

Name	Format	Description
HFSDIR-FILE-LENGTH UDLEN	PIC S(9) COMP	Length of the file. Undefined for Directory entries.
HFSDIR-FILE-DATE UDDAT	PIC S9(7) COMP-3	Date in YYMMDD format that file was last written to the HFS.
HFSDIR-FILE-TIME UDTIM	PIC S9(7) COMP-3	Time in HHMMSS format that the file was last written to the HFS.
HFSDIR-FMT UDRECFM	PIC X	Can be either “F” or “V” for fixed or variable format.
HFSDIR-FIX-LEN UDLRECL	PIC S9(4) COMP	If fixed format, the record length is found here.
HFSDIR-FILE-EXT UDXOF	PIC X(8)	The file extension is placed here. It is a copy of the extension, if any, from the name (below). It may not be present as the file may not contain an extension.
HFSDIR-CONTENT-SW UDTYP	PIC X	Will be either “A” for ASCII or “B” for binary.
HFSDIR-DIR-TYPE UDDIR	PIC X	Will be either “D” for directory or “T” for file.
HFSDIR-NAME UDNAM	PIC X(256)	The name of this entry.

HFS API Gotchas

It is unfortunately easy to get tangled up in HFS. It is also very simple to avoid these problems. You should read the following before attempting to use the HFS API.

1. Storage Creep

This is discussed above but worth repeating. HFS acquires approximately 11K of SHARED storage in CICS. 10K can go anywhere, above or below the line. 1K must be below the line. During the midst of a get or put process, an additional 4k of go anywhere storage is acquired. SHARED storage does not go away with the end of a CICS task but must be explicitly freed by the program that acquired it.

If you do not issue the *Close Access to HFS* request, your CICS DSA will slowly fill up with these little bits of SHARED storage. Therefore, in CICS, you really should code a HANDLE ABEND routine to close off the HFS API in the event of a program failure.

This is not a problem with the batch API as normal VSE end of job processing will clean up this memory.

2. Stray Records

A "stray record" is a record that HFS thinks is being used but is not contained in any file or directory chain within the HFS extent. Stray records can accumulate, and left untended can fill up an entire HFS extent.

Stray records occur when you neglect to issue the *End File* request following one or more *Put File* requests. This can range from one or two records to a great many depending on how large the file and how much of it got written. Neglecting to issue the *End File* requests is either a programming error, hopefully resolved during testing, or an abend situation.

In CICS, your handle abend routine should check to see if a *Put File* request is in progress, and if so, issue the *Backout File* request to clean up the HFS. It is harder to handle the abend issue in batch.

Periodically, you will need to run the RECOVER DEEP against your HFS extents. This will clean up HFS and release all stray records on the file. For your test files, you will probably need to run this often. For production data, the frequency will depend on the stability of your program code and is impossible to predict – monthly should suffice.

3. Locked Files

When you issue the first *Put File* request to an HFS file, the file is locked for write processing. This prevents any other user in any other partition in the VSE machine from accessing this file.

In a similar fashion, when you issue the first *Get File* request to an HFS file, the file is locked for read processing. Other tasks can also read it, but none may write it until all read activity has completed.

These locks are removed when

- An *End File* request is issued (either read or write),
- A *Backout File* request is issued for a write operation,
- A batch UNLOCK is run,
- The HFS is initialized, or
- VSE is IPLed.

In other words, if the locks are not handled at the proper time, in the abend handler, the locks can remain in effect for a long time.

In batch, this can be easily remedied by constructing your JCL like:

```
* $$ JOB …
// JOB
  .  .  .     Your job statements here
/&
// JOB UNLOCK
  .  .  .      DLBL and EXTENT as needed
// LIBDEF *,SEARCH=( . . .
// EXEC CSIHFBAT
 ACCESS hfsname
 UNLOCK
/*
/&
* $$ EOJ
```

Here, an extra VSE JOB is added to the end of the POWER JOB. This JOB ensures that locks are reset for the current partition and the indicated HFS extent. You can repeat the ACCESD and UNLOCK commands in pairs as many times as needed to be certain that all HFS extents used in the jobstream are unlocked.

You can and should add the same VSE JOB to your CICS JCL as well. This will clean up the HFS in case CICS tumbles, but will not remove locks resulting from transaction failures unless, of course, you are willing to cycle CICS to clean them up.

In an emergency situation, locks can be removed for other than the partition the CSIHFBAT job is run in, but to paraphrase Elmer Fudd, "*Be vewy vewy caweful hewe.*"

```
. . .
// EXEC CSIHFBAT
 ACCESS hfsname
 RELEASE F2
/*
```

This sequence of commands will release the locks for partition "F2." This will release **all** of the file locks for F2 – there is no way to detect whether a lock is currently valid or the result of some preceding failure. If there are legitimate locks currently active in CICS and you run the RELEASE command you run the risk of storage violations and or CICS crashes resulting from the RELEASE command.

The best recourse in CICS is to include properly coded HANDLE ABEND routines in the transactions that use the HFS API.

HFS API HANDLE ABEND Routine

The following steps outline your HANDLE ABEND routine for the HFS API.

1. If the HFS API Token is active (not equal to binary zeroes) continue, otherwise no further processing is required.
2. If in the midst of a *Put File* request (determined by examining the function code in the HFS API Parameter Area, or by other means as necessary) then issue the *Backout File* request and proceed to step 4 below.
3. If in the midst of a *Get File* request issue the *End File* request.
4. Finally, issue the *Close Access to HFS* request to clean up storage.

Following these steps should avoid the gotchas mentioned above.

If your task uses one or more EXEC CICS LINKs then each LINK level may need to contain a HANDLE ABEND routine to properly clean up the HFS in the event of a transaction abend. This is only a problem when the Token remains open when the LINKed to program returns to its caller.

The *Close Access to HFS* requests assists here by zeroing out the Token once the HFS API is closed. Your lower LINK level program need only pass this updated Token to its caller.

In the event that an abend occurs within CSIHFAPI itself, a HANDLE ABEND routine is already provided in accordance with the steps outlined above. It will zero the token as a result of this processing.

HFS API Test Program – CSIHFAPT

This program was developed to test the HFS API in a CICS environment. You may find it useful to try the various API requests here prior to coding them into your application. The necessary RDO commands are shown in "Installation: Step 5 – CICS Table Additions."

When you enter the "HFST transaction, you will be presented with a screen that looks like:

```
CSIHFAPT                    Hierarchical File System              03/21/06
                                 API Test Program                 15:28:56
------------------------------------------------------------------------------
   CMD: _                                          A - Open HFS Access
   SEP: /                                          B - Open File
  FILE: _______                                    C - Get File
  OPT1: _                                          D - Put File
  OPT2: _                                          E - End File
 NAME1: ________________________________           F - Delete File
 NAME2: ________________________________           G - Rename File
  DATA:                                           H - Change Dir
                                                  I - Get Current Dir
  RESP:                                           J - Make Dir
                                                  K - Get Dir (1st)
                                                  L - Get Dir (next)
                                                  M - Backout File
                                                  Y - Option(s)
                                                  Z - Close HFS Access
```

The eight entry fields correspond directly to fields in the HFS API Parameter list:

CSIHFAPT Screen	HFS API Parameter List
CMD	HFSAPI-FUNC
SEP	HFSAPI-SEP
FILE	HFSAPI-HFS-NAME
OPT1	HFSAPI-OPTBYTE1
OPT2	HFSAPI-OPTBYTE2
NAME1	HFSAPI-NAME1
NAME2	HFSAPI-NAME2
DATA	HFSAPI-BUFFER

The NAME1 and NAME2 fields are abbreviated to 32 bytes to fit the 3270 screen conveniently. Likewise, the DATA field is only 32 bytes long. You can, however, read and./or write longer files by repeatedly issuing the *Get File* or *Put File* requests.

The transaction will terminate when you press the CLEAR key.

Installation

Step 1 – UNZIP

Unzip the HFSJCL.ZIP file in the installation directory. The resulting CSIHFS.BJB file is in EBCDIC, with 80-byte records and no CRLF.

Step 2 - Upload Distribution Jobstream

Upload the distribution jobstream, CSIHFS.BJB, to the POWER RDR queue. File is in EBCDIC, with 80 byte records and no CRLF.

Step 3 - Release Install Job

Alter the POWER job (CSIHFS) that was loaded in the RDR Queue to run in a partition that has labels established for link editing.

```
A RDR,CSIHFS,DISP=D,CLASS=?
```

Step 4 - Supply LIBDEF

When job CSIHFS starts up, it will pause allowing you to enter a "// SETPARM" statement for the *library.sublibrary* that you want to contain HFS.

```
*    // SETPARM SUBLIB='LIB?.SUBLIB?'
*
// PAUSE   ENTER ABOVE SETPARM AS REQUIRED
```

When the installation job is complete, review the printed output for errors. Common problems that may occur during the installation job are:

- Your library is full.
- The SETPARM statement was omitted or improperly declared.
- Install job was run in a partition not set up for Link Editing.

Step 5 – CICS Table Additions

In order to use the HFS online component you need to add three programs and one transaction to CICS. Use the following RDO definitions as a guide.

```
DEFINE PROG(CSIHFI10) GROUP(HFIL) LANG(A) EXECKEY(CICS)
DEFINE PROG(CSIHFI20) GROUP(HFIL) LANG(A) EXECKEY(CICS)
DEFINE PROG(CSIHFI30) GROUP(HFIL) LANG(A) EXECKEY(CICS)
DEFINE TRANS(HFIL)    GROUP(HFIL) PROG(CSIHFI10) TASKDATAKEY(CICS)
```

If you will be using the CICS API, then you need to add the following as well:

```
DEFINE PROG(CSIHFAPI) GROUP(HFIL) LANG(A) EXECKEY(CICS)
DEFINE PROG(CSIHFAPS) GROUP(HFIL) LANG(A) EXECKEY(CICS)
```

If you want to use the CICS API Test Program, add:

```
DEFINE PROG(CSIHFAPT) GROUP(HFIL) LANG(A) EXECKEY(CICS)
DEFINE TRANS(HFST)    GROUP(HFIL) PROG(CSIHFAPT) TASKDATAKEY(CICS)
```

You can name the transaction anything you want to.

It is critically important that the EXECKEY (for programs) and TASKDATAKEY (for the transaction) be set to "CICS" for *CICS Transaction Server*. Failure to do so may result in an abnormal termination of CICS.

Error Messages

CSIHFLOD Messages

CSIHFLOD-001 E ERROR DETECTED - PROCESSING TERMINATED

One or more errors were detected by CSIHFLOD. The actual error should precede this message. These are probably errors returned from HFS access. Error(s) need to be corrected before re-running the job.

CSIHFLOD-003 E LIBR LIBRARY.SUBLIB MISSING

The library,sub-library name on the LIBR command must be corrected.

CSIHFLOD-004 E LIBR FILE NAME MISSING – PROCESING TERMINATED

The file name portion of the LIBR command must be corrected.

CSIHFLOD-005 E TO FILE NAME MISSING – PROCESSING TERMINATED

Most likely the TO segment was omitted. Correct the command and resubmit the job.

CSIHFLOD-006 E BLOCKSIZE INVALID

The BLOCKSIZE operand was found to be invalid. Correct the command and resubmit the job.

CSIHFLOD-007 E LRECL INVALID

The LRECL operand was found to be invalid. Correct the command and resubmit the job.

CSIHRCV Messages

These messages are generated during the RECOVERY process.

CSIHFRCV-01 RECORD TYPE INVALID, FILE ASSUMED

This message is informational and is followed by the name of the file where the error is detected. Processing continues with the assumption that this is a reference to an HFS file. Other errors may occur in this situation. If DEBUG is specified on the RECOVER command, the directory entry for this file will be dumped for analysis.

CSIHFRCV-02 CROSS-LINKED FILE FOUND

This message occurs when a file or directory record is referenced more than once in the HFS directory. The message is followed by two file names. The first is the name of the first file that was encountered with this record pointer, and the second is the name of the current file being investigated by the recovery process. In all recovery types, the second entry will be removed from the HFS Directory.

This message is considered severe. It will be written to both the report and the system console. It will only appear on the console once, but may appear several times in the Recovery Report. For both QUICK and STANDARD recovery, CSIHFRCV will attempt to upgrade the recovery process to DEEP.

CSIHFRCV-03 SERIOUS ERROR DETECTED, DO YOU WISH TO RECOVER (YES/NO)

Whenever a severe error is encountered, and recovery is either QUICK or STANDARD, CSIHFRCV will attempt to upgrade the recovery process to DEEP. If AUTO was specified on the RECOVER command, this process will take place immediately. If NOAUTO was specified, CSIHFRCV will prompt the console operator prior to upgrading the recovery level. If the operator responds “NO” or NOMSG was also specified, recovery operations will not be upgraded.

CSIHFRCV-04 UPGRADING TO DEEP RECOVERY

This message is written to both the console and the recovery report whenever CSIHFRCV upgrades the recovery level. Recovery level is always upgraded to DEEP regardless of where it started from.

CSIHFRCV-05 CANNOT LOCATE FILE HEADER

Each file begins with a File Header record. This record could not be located (probably due to a bad pointer reference in the directory). This message is followed by the file name causing the error. In all cases, the offending directory entry will be removed from the HFS file.

CSIHFRCV-06 UNEXPECTED FILE TYPE ENCOUNTERED, SHOULD BE *Xxxxx*

This message occurs when the directory entry points to the wrong kind of file (for example, a directory chain points to a file record). In all cases, the directory reference will be removed from the HFS file. The expected record type is shown in the body of the message. This message is always followed by the name of the file from the directory which caused the error.

This message is considered severe. It will be written to both the report and the system console. It will only appear on the console once, but it may appear several times in the recovery report. For both the QUICK and STANDARD recovery modes, CSIHFRCV will attempt to upgrade the recovery process to DEEP.

CSIHFRCV-07 ID RECORD CORRUPTED, CANNOT RECOVER

The initial File Id record has become corrupted. Recovery is not possible. Reload the file from your most recent backup. This message is always followed by a hexadecimal dump of the File Id record for problem analysis and troubleshooting.

CSIHFRCV-08 FILE LINKED TO *xxxx* RECORD, HFS FILE INTEGRITY CANNOT BE ASSURED

This message is detected during either STANDARD or DEEP recovery. It indicates that a file is pointed at one of the forbidden records. Forbidden records are the System File Id record (0), one of the FAT records, one of the Extent records, or the ROOT directory. In all cases, the associated directory entry will be removed.

This is a serious error; the integrity of the HFS file cannot be assured. The only safe way to proceed following this error is to reload the HFS file from your most recent backup. It is not possible to determine whether the file is intact and usable.

CSIHFRCV-09 FILE INCOMPLETE - FILE TRUNCATED

This message is detected by DEEP recovery only. Recovery stopped short of the number of file records indicated in the allocation table for the individual file. The file will be truncated to what was found in the HFS. The file length in the directory will be adjusted accordingly.

CSIHFRCV-10 ONE OR MORE FILE RECORDS

This message is detected by DEEP recovery only. The individual file's allocation table has been corrupted somewhere in the middle of the table. Recovery will delete the file as it is currently unusable.

CSIHFRCV-11 FILE CROSS-CHAINED TO ANOTHER FILE - FILE DELETED

This message is detected by DEEP recovery only. While verifying the file, a pointer in the file's allocation table pointed to a record already associated with another file. The Recovery process will determine which of the two files is the correct owner of the affected record, and delete the other.

CSIHFRCV-12 FILE LINKED TO PROHIBITED RECORD – FILE DELETED

This is similar to message 11 above, only this case the file is linked to one of the system-prohibited records (File Id, FAT, EXTENT and ROOT Directory records). The file will be deleted.

This is a serious error; the integrity of the HFS file cannot be assured. The only safe way to proceed following this error is to reload the HFS file from your most recent backup. It is not possible to determine whether the file is intact and usable

CSIHFRCV-13 FILE CHAIN CORRUPTED (*xxxxx* RCD DETECTED) - FILE DELETED

This indicates essentially the same problem as message 12; it has a different number for tracing and debugging purposes.

CSIHFRCV-14 FILE CHAIN INCOMPLETE - FILE DELETED

This message is detected by DEEP recovery only. An individual file's allocation table may extend over one or more 4K records on the HFS. Here the second or subsequent record could not be read from the HFS. This message should not occur. In the event it does, it will probably be preceded by errors from the Access Modules.

CSIHFRCV-15 ONE OR MORE ERRORS DURING *xxxxxxxxxxx*

In order to provide a readable control report, recovery is a two-step process. First the errors are detected, and second, any deletions or corrections are made. During this second step one or more errors occurred. This message should not occur. In the event it does, it will probably be preceded by errors from the Access Modules.

CSIHFRCV-16 RECORD NOT IDENTIFIED IN FAT – FAT CORRECTED

A record was retrieved from the HFS and its corresponding position in the FAT was set to free. The FAT is corrected *immediately* when this message appears (this is the only exception to the two-step processing described in message 15 above).

This situation can arise in all recovery modes; however, the FAT is not guaranteed to be correct unless you run a DEEP recovery against the HFS file.

CSIHFRCV-17 CONSOLE RESPONDED "NO" TO UPGRADE QUERY

Informational message placed in the control report when the console operator responds “NO” to the upgrade prompt (message 03 above).

CSIHFRCV-18 DIRECTORY RECORD NOT FOUND

Informational message describing nature of error. This will probably be followed by other messages and possibly a wholesale deletion of data from the file.

CSIHFRCV-19 CANNOT RECOVER WITH JOURNAL/CACHE ACTIVE

Recovery cannot be run concurrently with either journaling or cache processing active.

CSIHFRCV-20 SCHEDULE DEEP RECOVERY FILE=*xxxxxxx*

This message is generated when NOUPDATE was specified and an error was detected that requires a DEEP recovery to correct. You should schedule a DEEP recovery at the earliest convenience to correct the file.

CSIHFSDX Messages

These messages are generated during the edit operation on File Definitions, Conversion Rules and Translate Tables.

CSIHFSDX-001 E FILE NAME MISSING OR INVALID

Correct the File Name and resubmit the job.

CSIHFSDX-002 E RECFM OPERAND MISSING

RECFM operand is required. Correct the command and resubmit the job.

CSIHFSDX-003 E RECFM OPERAND INVALID

Correct the RECFM operand and resubmit the job.

CSIHFSDX-004 E COMMAND OUT OF SEQUENCE

This error most likely arises from a missing END command from a preceding FILE or CONVERT command. Correct the command sequence and resubmit the job.

CSIHFSDX-005 E COMMAND INVALID FOR xxxxxxxx PROCESS

A command available for CONVERT was located within a FILE sequence, or a command specific to FILE was located in a CONVERT sequence. Correct the command sequence and resubmit the job.

CSIHFSDX-006 W UNEXPECTED GAP IN OFFSETS, EXPECTED XXXXX

This warning message is issued whenever the offsets in a File Definition are not in strict numerical order. Only you can determine whether the warning is relevant or not.

CSIHFSDX-007 E FIELD NAME MISSING OR INVALID

A FIELD command must have name associated with it. Correct the command and resubmit the job,

CSIHFSDX-008 E PIC AND FMT ARE MUTUALLY EXCLUSIVE

You cannot use both PIC and FMT to define a FIELD. Correct the command and resubmit the job.

CSIHFSDX-009 E EITHER PIC OR FMT MUST BE SUPPLIED

You must supply either a PIC or a FMT for a FIELD definition. Correct the command and resubmit the job.

CSIHFSDX-010 E PIC INVALID OR UNSUPPORTED

The PIC value you entered is invalid or unsupported (see FIELD definition above). You may have to use the FMT operand instead of the PIC operand for this FIELD. Correct the command and resubmit the job.

CSIHFSDX-011 E TABLE COMMAND INCOMPLETE - OCCURS REQUIRED

The TABLE command must specify the OCCURS (or maximum OCCURS) value. Correct the command and resubmit the job.

CSIHFSDX-012 W RECORD LENGTH WARNING, CALCULATED 99999

The sum of the offsets and field lengths differ from the record length given in the FILE command. Review the File Definition to ensure that no fields are missing and all fields are defined with their proper lengths. It may be necessary to use the LIST command to determine how CSIHFSDX interpreted your input.

CSIHFSDX-013 verify warning messages

There are a variety of warning message issued when the preceding command is verified. These messages should be self-explanatory.

CSIHFSDX-014 E FILE DEFINITION NAME MISSING OR INVALID

The CONVERT command must reference a previously defined File Definition. Correct the command and resubmit the job.

CSIHFSDX-015 E DECIMAL SPECIFICATION INCORRECT, …

Currently only two specifications are allowed for the DECIMAL operand, COMMA and PERIOD. If you need another, contact CSI Technical Support for Assistance.

CSIHFSDX-016 E FIELD NAME REQUIRED ON MOVE COMMAND

The MOVE command must reference a field defined in the relevant File Definition by name. Correct the command and resubmit the job.

CSIHFSDX-017 E LITERAL VALUE MISSING

Correct the command and resubmit the job.

CSIHFSDX-018 E ADD COMMAND FORMAT INCORRECT

Refer to the description of the ADD command above to correct the command and resubmit the job.

CSIHFSDX-019 E IF STATEMENT INCOMPLETE

Refer to the description of the IF command above to correct the command and resubmit the job.

CSIHFSDX-020 E FIELD AND LITERAL ARE MUTUALLY EXCLUSIVE

Comparisons are a field to either another field or a literal, not to both. Correct the command and resubmit the job.

CSIHFSDX-021 E EITHER FIELD OR LITERAL MUST BE SUPPLIED

You must supply either a FIELD or a LITERAL to complete the comparison. Correct the command and resubmit the job.

CSIHFSDX-022 E THEN LABEL MISSING

The IF command must supply a label. Correct the command and resubmit the job.

CSIHFSDX-023 E LABEL NAME MISSING

A label is necessary on the GOTO command. Correct the command and resubmit the job.

CSIHFSDX-024 E MORE THAN 32 TABLES DEFINED

The run time component of File Conversion can only handle 32 tables total. Expanding this limit will require source-level changes to **INTERCEPTOR**. Contact CSI Technical Support for assistance.

CSIHFSDX-025 E TABLE NAME MISSING

You must supply a unique name for each TABLE. Correct the command and resubmit the job.

CSIHFSDX-026 E TRANSLATE FROM OPTIOIN MISSING

You must supply a FROM operand on the TRANSLATE command. Correct the command and resubmit the job.

CSIHFSDX-027 E TRANSLATE TO OPTION MISSING

You must either supply a complete TO translate table (see "Appendix D – Translate Tables") or specify the REVERSE operand on the TRANSLATE command.

CSIHFSDX-028 E TRANSLATE from LINE 99 INVALID DATA

Invalid hexadecimal data was discovered on the indicate line in the indicate FROM or TO table. Valid hexadecimal data is 0-9, A-F. Correct the command and resubmit the job.

CSIHFSDX-029 E TRANSLATE TABLE NAME MISSING OR INVALID

You must supply a name for this TRANSLATE table. Correct the command and resubmit the job.

CSIHLBLO Messages

These messages are generated by the open intercept process. These messages will be displayed on the system console.

CSIHLBLO 001 E CANNOT ACCESS HFS JOURNAL PARTITION

The HFS Journal process must be active in order for HFS File Interception to work properly – see "System Requirements". Start the Journal process and resubmit the job.

CSIHLBLO 002 E CANNOT LOAD CSIHLMOD

The VSE library containing HFS phases must be accessible in each partition that will be using HFS. Correct your JCL and resubmit the job.

CSIHLBLO 010 I OPEN OF xxxxxxx – ACCEPTED

Informational message displayed on the console whenever HFS File Interception intercepts I/O for a file.

CSIHLMOD Messages

These messages are generated by the I/O module replacement, CSIHLMOD. These messages will be displayed on the system console.

CSIHLMOD-002 E WRONG LENGTH RECORD ERROR 99999/99999

This message occurs only if the current record size is greater than the maximum specified on the FILE command in effect for this job. The File Definition will need to be revisited before you can continue. Both the maximum record size and the actual record size are displayed in the message to assist you in correcting the File Definition.

CSIHLMOD-003 E INTERNAL ERROR ON xxxxxxx CANNOT CONTINUE

Additional messages will accompany this message that further define the error. These will either be from HFS (1xx, 2xx, 3xx or 4xx) or from the File Conversion (6xx). Please have all messages generated available when contacting CSI Technical Support for assistance.

CSIHLMOD-004 E CANNOT LOAD CSIHFCVT

The VSE library containing HFS phases must be accessible in each partition that will be using HFS. Correct your JCL and resubmit the job.

HFBAT Messages

These messages are placed on the console during forward recovery. They each require a console response.

HFBAT01 I HFS not restored using PHYSICAL option. Continue? (Y/N)

The HFS was restored using NOPHYSICAL (logical) backup tape. It is unlikely that forward recovery will function properly in this situation.

HFBAT02 I Journal begins after backup. Do you wish to continue? (Y/N)

There is a gap in time between the current state of the HFS file following a restore and the first Journal tape record encountered. Ensure that you have the proper tape mounted for the recovery process.

HFBAT03 I Journal tape completed. Is there another tape (Y/N)?

At end of file for each journal tape you will be prompted for more input. Respond "Y" to mount a new volume,. Respond "N" when all volumes have been processed.

HFSJ Messages

Journaling produces the following messages. These messages are written to the VSE console and to the SYSLST output of the journal process.

HFSJ101E CANNOT ALLOCATE GETVIS FOR CSIHF00

This error should not occur. It indicates insufficient 24-bit System GETVIS.

HFSJ102E CSIHF00 NOT IN SVA

This error should not occur. Contact CSI Technical Support for assistance.

HFSJ103E UNABLE TO LOAD CSIHF00 RC=*xx*

This error should not occur. Contact CSI Technical Support for assistance.

HFSJ104E LESS THAN 256K 24-BIT GETVIS

This error should not occur. Contact CSI Technical Support for assistance.

HFSJ105I *nnnnn* QUEUE ENTRIES GENERATED

Informational message indicating the number of buffers generated for journaling purposes. The program currently attempts to obtain 1,024 buffers.

HFSJ106I HFS JOURNAL OPEN ON HFSJRNA/B

Informational message that indicates which journal file is currently in use. Message is written when starting the journaling process.

HFSJ107I HFS JOURNAL HFSJRNA/B CLOSED

Informational message indicating that a journal file has been closed. This message appears when the journal is switched and when the journaling partition is terminated.

HFSJ108I HFS JOURNAL SWITCHED TO HFSJRNA/B

Informational message which indicates that a Journal Switch occurred. Journal output is now being directed to the file named in the message.

HFSJ110I - ENTER HFS JOURNAL COMMAND

Console prompt that appears in response to a MSG *xx* request.

HFSJ111W CANNOT COMPLETE HFS CACHE REQUEST F=*xxxxxxx*

Program was unable to completely acquire storage for the cache. The size of the cache will be shown in a subsequent message.

HFSJ112I – <*trace messages*>

Various self-explanatory trace messages appear under this number.

HFSJ113E – JOURNAL ALREADY ACTIVE

An attempt was made to start HFS journaling while it is currently active in another partition. Only one journal partition is allowed. If this message was issued as a result of an earlier abend in the journal partition, the error can be overridden by adding the FORCE option to the JOURNAL request. This command line has the following syntax:

JOURNAL (BUFFERS(*nnn*) FORCE)

HFSJ114E – CANNOT ALLOCATE HLBL AREA

Program was unable to allocate the HLBL area. The CSI File Interception Facility will not work until this problem is corrected. Either run Journaling in a bigger partition, or adjust other buffers so that the HLBL area can be created.

Internal Errors

Error messages generated by HFS are prefixed by a three-digit number. Message types are based on number range and have the following categories:

- 100s – Parameter errors (These represent programming errors and it is unlikely you will ever see them.)
- 200s – Processing errors
- 300s – Internal errors that require updates to the software. (These messages are severe and should be reported to CSI Technical Support.)
- 400s – Warning messages
- 600s – Issued by File Conversion Process.

For the most part, these messages are believed to be self-explanatory.

1xx Messages

These messages represent internal parameter errors and should never be seen.

101 - INVALID FUNCTION CODE
102 - FUNCTION NOT IMPLEMENTED YET
103 - FILE NAME MISSING
104 - COMMAND SEQUENCE ERROR
105 - NO RESPONSE BUFFER IN PARM
106 - DIRECTORY NAME MISSING

2xx Messages

These messages are generated during normal operations of HFS. These messages may issued in response to one of the HFS commands described in this manual. In these cases the message refers to the immediately preceding command and should be self-explanatory in context.

You may also see them following, or part of, other error messages generated by other programs which are written to use HFS. In these cases you should refer to the error message from that product to determine how to proceed with the problem.

If all else fails, contact CSI Technical Support for assistance.

201 - HFS NOT OPEN
202 - GETVIS REQUEST FOR XXX BYTES FAILED RC=
203 - UNABLE TO ACCESS FILE=
204 - UNABLE TO LOAD CSIHF00 RC=
205 - ANCHOR TABLE LIMIT EXCEEDED
206 - BUFFER TOO SHORT
207 - FILE IS LOCKED
208 - INVALID DIRECTORY NAME
209 - PATH NOT FOUND
210 - DUPLICATE FILE
211 - NON-DIRECTORY IN PATH

212 - INVALID PATH
213 - NO SPACE IN FILE RC=
214 - PATH NAME NOT DIRECTORY
215 - FILE NAME MISSING OR INVALID
216 - ACCESS IN MIDST OF WRITE
217 - FILE NOT FOUND
218 - FILE NAME NOT DATA FILE
219 - DIRECTORY NOT EMPTY
220 - UNKNOWN ALIAS PROCESS
221 - ERROR DETECTED IN MODULE
222 - FILE OPENED READ ONLY
223 - INVALID LOGICAL UNIT
224 - RBA REQUEST BEYOND FILE SIZE RBA=
225 - FILE RECOVERY IN PROGRESS, ACCESS DENIED
226 – SEEK INVALID FOR EXTERNAL FILES
227 – INVALID TOKEN
228 – PRODUCT AUTHORIZATION HAS EXPIRED

3xx Messages

These messages represent serious internal problems with HFS. A series of snap dumps will be written to SYSLST when these messages occur. Have these snap dumps available when contacting CSI Technical Support for assistance.

301 - INVALID SEEK ADDRESS
302 - I/O ERROR
303 - RECORD NOT DIRECTORY
305 - DIRECTORY CHAINING ERROR
306 - FILE CHAINING ERROR
307 - INVALID INITIAL EXTENT CANNOT FORMAT
308 - VOLSER MISSING CANNOT FORMAT
309 - EXTENT RECORD INVALID OR MISSING CANNOT CONTINUE
310 - CANNOT ASSIGN TO VOL=xxxxxx |RC=xx
311 - NO EXTENT FOR RCD=xxxxxxxx F=filename
312 – EXTENT IN USE FILE=xxxxxxxx
313 – DIETFAT ERROR R=xxxxxxxx
314 – NO DLBL FOR FILE

4xx Messages

These messages are warnings. HFS processing continues. These messages should be intercepted and dealt with internally by HFS and those programs using HFS.

401 - UNABLE TO DELETE COMPLETELY
402 - FILE OPENED READ ONLY
403 – SEEK OUTSIDE OF FILE LIMITS
404 - JOURNALLING NOT ACTIVE
405 - NO JOURNAL ALET

6xx Messages

These messages are generated by the File Conversion process when attempting to convert a file. They should not occur as the underlying problem should have been detected when the Conversion Rules were created. If these messages occur, an attempt was made to run conversion processing with a error-filled set of Conversion Rules.

These messages should be accompanied by message CSIHLMOD-003 (see above) and in all cases, processing is terminated.

The messages should be self-explanatory.

601-FIELD UNDEFINED: field name
602-LABEL UNRESOLVED: label name
603-IF UNRESOLVABLE

Appendix A – Parameter Syntax

The syntax of HFS commands is similar to the syntax used by the IBM utility IDCAMS:

- Comments can be inserted on one or more lines by embedding them between "/*" and "*/" pairs.
- An entire line can be defined as a comment line by keying a semicolon ':' as the first non-blank character in the line.
- Input lines are parsed from columns1 thru 72
- Commands can be continued on another line by adding a dash (a hyphen) to the right of the command text, as in the following example:
- An equals sign, "=", may also be used in the case where a single operand is enclosed in parenthesis, for example HFS(HFS01) and HFS=HFS01 will both be parsed correctly.
- Either blanks or commas can be used to separate Operands from the Command and from other Operands.
- Character data must be enclosed in single-quotes if it contains either imbedded spaces, comma, parenthesis, or the slash. "/", character. Otherwise the quote marks are optional.

Comments can be freely interspersed with the command, such as:

```
HLBL TESTFIL                  /* DLBL NAME */ -
    '/ACCOUNTING/DAILY/TEST'  /* FILE NAME IN HFS */ -
      0                       /* RETENTION */ -
      SD                      /* FILE TYPE */ -
      HFS=HFS01               /* TO  H F S 0 1 */
```

Commands listed in this document use the following syntax conventions:

- Words in UPPER CASE are required and must be typed as shown in the command description.
- Words in lower case, such as file name, need to be replaced by information meaningful to you.
- Where there are options, the default value, if any, is underlined.
- Alternative required keywords are enclosed by curly braces '{ }'. Choices are separated by a vertical bar ' | '.
- Optional keywords are enclosed by brackets '[ ]'.
- Short aliases of commands or keywords are listed below the command line.

Appendix B – Undefined Files

RECFORM=UNDEF files may or may not work with The HFS File Interception Facility. It all depends on what the underlying data organization truly is. The following discussion was written by the developer and may get a bit technical.

The HFS File Interception Facility ignores the block size specification in the DTF or COBOL program. For all practical purposes, sequential files written to an HFS extent by HFS consist of one great big block – the file size.

This lack of blocking is of no consequence to well-behaved assembler and COBOL programs that use VSE LIOCS for their file I/O. These programs expect to write and read logical records and do not need to care how many of them are grouped together into a larger block of data. A well-behaved COBOL or assembler program shouldn't care.

In VSE you can write a file using a DTF RECFORM=UNDEF and subsequently read it with a RECFORM=FIXxxx or RECFORM=VARxxx DTF provided that your user written program blocked it correctly in the first place. Likewise, you can write a RECFORM=FIXxxx or RECFORM=VARxxx file and subsequently read it using a DTF RECFORM=UNDEF provided you are willing to take on the responsibility of properly de-blocking the file yourself when you read it.

VSE gets away with this by simply writing however many bytes are supplied on the output RECFORM=UNDEF DTF, and when reading, attempting to read a maximum block and determining from the CCW how much data was actually retrieved. For VSE this is a piece of cake.

Since, as stated above, The HFS File Interception Facility does not preserve block size when writing data to the HFS extent, it cannot read a RECFORM=UNDEF file in complete emulation of VSE LIOCS.

This is primarily a problem for input RECFORM=UNDEF files only. However, a file written as RECFORM=UNDEF but blocked properly for a subsequent read as RECFORM=VARxxx will not work either.

Trying to handle this internally is complicated by the fact that there is no identifiable difference in the DTF itself between a RECFORM=UNDEF and RECFORM=VARUNB file – not even one stinking bit. The subtleties between the two DTFs are handled at assembly time.

You can attempt to get around this with the **UNDEF** operand of the **HLBL** statement, but this will only help a tiny bit. Assuming that you have a generic backup utility that reads files as RECFORM=UNDEF, then adding the UNDEF operand will let the utility program read it. However, the read will always respond with a maximum block size of data. Consequently, a file backed up this way can only be reloaded into an HFS File Interception Facility controlled HFS extent.

The best option is: don't try to use The HFS File Interception Facility on files written with or read by a RECFORM=UNDEF DTF.

Appendix C – CRLF Processing

If you want to process CRLF delimited files directly in your program(s), The HFS File Interception Facility can provide some assistance for you. The processing described below occurs if the **CRLF** operand is included on the **HLBL** command.

Define your file:

Assembler	COBOL

`DTFSD RECFORM=VARUNB`	`FD  name`
	`    RECORD CONTAINS 0 TO ??? CHARACTERS`
	`    BLOCK  CONTAINS 1 RECORD`
	`    DATA RECORD IS DREC.`
	`01  DREC.`
	`    05  DATA  PIC X OCCURS 0 TO ??? TIMES`
	`                    DEPENDING ON 77-COUNT.`
	(or some other field in WORKING-STORAGE)

Write the file

Assembler	COBOL
(This depends on other settings in the DTF. You will, however, need to establish a proper LLBB at the beginning of the logical record.)	`MOVE stringlength TO 77-COUNT.` `WRITE name FROM data.area.in.working.storage.`

The HFS File Interception Facility will do a couple things here. It will strip off the leading LLBB provided by COBOL (or created by your assembler code). It will also ensure that the line ends in a CRLF sequence, and if you do not supply it, HFS will add the CRLF for you.

Read the file

Assembler	COBOL
The data record returned will include an LLBB as the first four bytes of the record area. The length portion includes the 4 byte length of the LLBB itself.	COBOL does you a favor(?) by stripping off the LLBB and presenting you with only the data component of the record. You are probably stuck scanning the record for the CRLF sequence to terminate your parsing.

The HFS File Interception Facility scans the data stream looking for the CRLF sequence and a variable length record to you including the trailing CRLF. The record is returned to you as if it was obtained from a variable-length-unblocked file to be compatible with COBOL processing.

HFS does not provide assistance in parsing the string, but it will de-block the file eliminating some complexity in your code.

Appendix D – Translate Tables

You can, if necessary, create your own translate tables for use in the File Conversion process. You can have as many translate tables as needed. Each table becomes a separate file within the HFSGEN HFS.

To utilize your own translate table, you need to code the TRANSLATE operand on the CONVERT command when creating Conversion Rules.

Translate tables are maintained using program CSIHFSDX – the same program that you use to maintain File Definitions and Conversion Rules (see above). Each translate table used in The HFS File Interception Facility actually consists of two tables:

FROM – for conversion from EBCDIC to ASCII, and
TO – for conversion from ASCII to ABCDIC.

The FROM table is used when converting from mainframe data to a string. The TO table is used when converting a string to mainframe data.

A single command is used to create the table:

TRANSLATE 'name' REVERSE | **NOREVERSE –**
FROM(hex data) –
TO(hex data)

Name	This is the name for the translate table. This name is specified in the TRANSLATE operand of the CONVERT command. The name can be from 1 to 248 characters in length.
REVERSE|**NOREVERSE**	Specify REVERSE if you want the program to generate the second table, NOREVERSE if you will be generating it yourself. NOREVERSE is the default.
FROM(hex data)	This defines the 256 byte translate table for conversion from EBCDIC to ASCII. This table is used when converting data from mainframe format to CRLF delimited string, or when the Sequential File is opened for output. The example below should make the format obvious.
TO(hex data)	This defines the 256 byte translate table for conversion from ASCII to EBCDIC. This table is used when converting data from CRL delimited string to mainframe format, or when the Sequential File is opened for input. The example below should make the format obvious.

The example below recreates the standard or default translation tables used by **INTERCEPTOR**.

```
/* ***************************************** */
/*   THIS IS THE STANDARD TRANSLATE TABLE USED    */
/*   BY INTERCEPTOR TO TRANSLATE FROM EBCDIC      */
/*   TO ASCII AND FROM ASCII TO EBCDIC            */
/* ***************************************** */
TRANSLATE 'ASCII STD' -
                     FROM( -
                          00010203EC09CA1CE2D2D30B0C0D0E0F  -
                          10111213EFC508CB1819DCD81A1D1E1F  -
                          B7B8B9BBC40A171BCCCDCFD0D1050607  -
                          D9DA16DDDEDFE004E3E5E9EBB0B19E7F  -
                          20FF838485A0F28687A49B2E3C282B7C  -
                          268288898AA18C8B8DE121242A293BAA  -
                          2D2FB28EB4B5B68F80A5B32C255F3E3F  -
                          BA90BCBDBEF3C0C1C2603A2340273D22  -
                          C361626364656667686 9AEAFC6C7C8F1  -
                          F86A6B6C6D6E6F707172A6A791CE92A9  -
                          E67E737475767778797AADA8D4D5D6D7  -
                          5E9C9DFA9F1514ACABFC5B5DE4FEBFE7  -
                          7B41424344454647484 9E8939495A2ED  -
                          7D4A4B4C4D4E4F505152EE968197A398  -
                          5CF6535455565758595AFDF599F7F0F9  -
                          3031323334353637383 9DBFB9AF4EAC9  -
                         ) -
                       TO( -
                          00010203372D2E2F1605250B0C0D0E0F  -
                          10111213B6B5322618191C27071D1E1F  -
                          405A7F7B5B6C507D4D5D5C4E6B604B61  -
                          F0F1F2F3F4F5F6F7F8F97A5E4C7E6E6F  -
                          7CC1C2C3C4C5C6C7C8C9D1D2D3D4D5D6  -
                          D7D8D9E2E3E4E5E6E7E8E9BAE0BBB06D  -
                          798182838485868788899 19293949596  -
                          979899A2A3A4A5A6A7A8A9C04FD0A13F  -
                          68DC514243444748525 3545756586367  -
                          719C9ECBCCCDDBDDDFECFC4AB1B23EB4  -
                          4555CEDE49699A9BAB9F5FB8B7AA8A8B  -
                          3C3D626A646566202122 7023727374BE  -
                          7677788024158C8D8EFF061728299D2A  -
                          2B2C090AACADAEAF1B3031FA1A333435  -
                          36590838BC39A0BFCA3AFE3B04CFDA14  -
                          EE8F4675FDEBE1ED90EFB3FBB9EABD41  -
                         )
```

This table is included in a sample jobstream loaded into the VSE library during installation named CSIHFXLT.A. You can load this member into your favorite text editor to use as a starting point for your modifications.

The easiest way to create a translate table is to create the FROM table and let the program generate the TO table for you by specifying the REVERSE operand on the TRANSLATE command.

Even so, translate tables are tricky as you need to ensure that both representations for the byte are handled properly. Suppose, for instance, you started with the following table (also included in CSIHFXLT.A)

```
/* ********************************************** */
/*  YOU CAN USE THIS A STARTING POINT TO MAKE YOUR   */
/*  OWN TRANSLATE TABLE.  THIS TABLE MAKES NO        */
/*  CHANGES.                                        */
/* ********************************************** */
TRANSLATE 'NO TRANSLATION' REVERSE -
                      FROM( -
                           000102030405060708090A0B0C0D0E0F -
                           101112131415161718191A1B1C1D1E1F -
                           202122232425262728292A2B2C2D2E2F -
                           303132333435363738393A3B3C3D3E3F -
                           404142434445464748494A4B4C4D4E4F -
                           505152535455565758595A5B5C5D5E5F -
                           606162636465666768696A6B6C6D6E6F -
                           707172737475767778797A7B7C7D7E7F -
                           808182838485868788898A8B8C8D8E8F -
                           909192939495969798999A9B9C9D9E9F -
                           A0A1A2A3A4A5A6A7A8A9AAABACADAEAF -
                           B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF -
                           C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF -
                           D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF -
                           E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF -
                           F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF -
                          )
```

If you wish to change x'40' to x'20' (EBCDIC and ASCII space) you need to change the entry at x'40' into the table to x'20' **and** the entry at x'20' to x'40' in order to keep the translate table in synch with itself. Changes should always be done in pairs like this.