Hierarchical File System

A new file system, the Hierarchical File System (HFS), supports a directory structure analogous to that found on other UNIX platforms, and the integrated NFS server allows a distributed UNIX user to mount an OS/390 disk. It is implemented as part of the OS/390 DFSMS base component.

The Hierarchical File System is implemented in *containers* that are normal MVS data sets. A Hierarchical File System container has an MVS data set name with a DSNTYPE of HFS. The container data set may be handled by traditional DFSMS/MVS storage management tools, such as DFHSM or DFDSS. You decide which parts of the total Hierarchical File System are implemented in which containers. You may even implement the full Hierarchical File System in one container, or you may spread parts of the file system hierarchy over a number of individual container data sets.

DFHSM and DFDSS operates at the HFS container level, while the ADSTAR Distributed Storage Manager (ADSM) supports backup at the individual file level if you have the ADSM Unix System Services client installed.

The Hierarchical File System file system implements the typical characteristics of a UNIX file system:

- Files may have names up to 1023 characters in length.
- File names are case sensitive.
- File names may include special characters, such as spaces.
- Files are byte oriented as opposed to record oriented.
- Files are arranged in directories.

The Hierarchical File System is one of the important pieces that allows an OS/390 system to have the look and feel of a UNIX system.

The implementation of the Hierarchical File System is transparent to the type of the data to be stored. Data may be text or binary; the implementation does not care about this. However, you may need to know what kind of data you are working with when you exchange data with other systems.

> **Text Data:** In a client/server environment where MVS participates, you have to consider how to do ASCII-EBCDIC translation. At first this might seem to be only a matter of two translation tables, but it goes much further than that. As soon as you have to deal with different countries, you have to handle translations using the appropriate code page. OpenEdition provides in the shell environment a command (ICONV), and for programs coded in C a conversion routine iconv(). Both allow you to convert according to the code pages that are applicable.
>
> **Binary Data**: Binary data is much more complicated to handle than text data. Integer variables may be stored differently according to the byte order in storage (little endian (PC) versus big endian (S/390)). Different representations of floating point variables exist. There is no single simple solution for handling such differences. Each situation has to be analyzed to find a suitable solution.

In an ONC/RPC application as well as in a ECE/RPC application, the programming interface layer takes care of these translations and conversions.

To transfer files in and out of your OpenEdition system, you can use the FTPD server. With this server, you can transfer files directly in and out of the Hierarchical File system, as well as in and out of the traditional MVS data sets.

File archives of tar or pax format are very convenient to transfer files that are organized in a directory tree. The pax shell command allows you to expand the various kinds of archive file formats. It allows you to translate data from ASCII to EBCDIC or vice versa at the same time. The pax command to expand and translate a tar archive is the following:

pax -o from=ISO8859-1,to=IBM-1047 -x tar -rf your_tar_file.tar

Data exchange between traditional MVS data sets and the Hierarchical File System can be accomplished through the TSO/E copy commands that are provided with OpenEdition, OCOPY, OPUT, OGET, OPUTX AND OGETX.

OCOPY is based on DDNAME allocation. So you have to allocate your input and output DDNAMEs to MVS data sets or HFS files before you invoke the OCOPY command.
OGET, **OPUT**, **OGETX** and **OPUTX** give you the opportunity to specify the MVS data set name and HFS file name directly on the command invocation. OGETX and OPUTX support partitioned data sets, allowing you to copy all members of a partitioned data set in one command invocation. In addition you may apply a suffix to the new files. If, for example, you want to copy all members of your partitioned library **hlq.project.c** to the Hierarchical File System as files in a specific directory, you can have all member names of the form **membername.c** while you copy.

All commands support ASCII to EBCDIC conversion.

When you copy from a traditional MVS data set that contains binary data you should use the BINARY option of the OPUT or OPUTX command:

oput gzip.exe '/u/bossxen/gzip' binary

In case you copy a data set that contains text type data in ASCII encoding and you want to convert the data to EBCDIC while copying to the Hierarchical File System, use the command format shown below:

oput work.asc '/u/bossxen/work.asc' binary
CONVERT((BPXFX111))

The BINARY option is important, because it tells OPUT to handle the inout data set as a string of bytes, not as a collection of records.