24. Streams and Files 24.1

24.1. Opening and Closing File Streams 24.2

24.2. File Names 24.5

24.3. Incomplete File Names 24.9

24.4. Version Recognition 24.11

24.5. Using File Names Instead of Streams 24.13

24.5.1. File Name Efficiency Considerations 24.14

24.5.2. Obsolete File Opening Functions 24.14

24.5.3. Converting Old Programs 24.15

24.6. Using Files with Processes 24.16

24.7. File Attributes 24.17

24.8. Closing and Reopening Files 24.20

24.9. Local Hard Disk Device 24.21

24.10. Floppy Disk Device 24.24

24.11. I/O Operations to and from Strings 24.28

24.12. Temporary Files and the CORE Device 24.29

24.13. NULL Device 24.30

24.15. Deleting, Copying, and Renaming Files 24.31

24.16. Searching File Directories 24.31

24.17. Listing File Directories 24.33

24.18. File Servers 24.36

24.18.1. Pup File Server Protocols 24.36

24.18.2. Xerox NS File Server Protocols 24.37

24.18.3. Operating System Designations 24.38

24.18.4. Logging In 24.39

24.18.5. Abnormal Conditions 24.41

24. STREAMS AND FILES

Interlisp-D can perform input/output operations on a large variety of physical devices, including local disk drives, floppy disk drives, the keyboard and display screen, and remote file server computers accessed over a network. While the low-level details of how all these devices perform input/output vary considerably, the Interlisp-D language provides the programmer a small, common set of abstract operations whose use is largely independent of the physical input/output medium involved—operations such as *read*, *print*, *change font*, or *go to a new line*. By merely changing the targeted I/O device, a single program can be used to produce output on the display, a file, or a printer.

The underlying data abstraction that permits this flexibility is the *stream*. A stream is a data object (an instance of the data type **STREAM)** that encapsulates all of the information about an input/output connection to a particular I/O device. Each of Interlisp-D's general-purpose I/O functions takes a stream as one of its arguments. The general-purpose function then performs action specific to the stream's device to carry out the requested operation. Not every device is capable of implementing every I/O operation, while some devices offer additional functionality by way of special functions for that device alone. Such restrictions and extensions are noted in the documentation of each device.

The vast majority of the streams commonly used in Interlisp-D fall into two interesting categories: the *file stream* and the *image stream*.

A file is an ordered collection of data, usually a sequence of characters or bytes, stored on a file device in a manner that allows the data to be retrieved at a later time. Floppy disks, hard disks, and remote file servers are among the devices used to store files. Files are identified by a "file name", which specifies the device on which the file resides and a name unique to a specific file on that device. Input or output to a file is performed by obtaining a stream to the file, using **OPENSTREAM** (page 24.2). In addition, there are functions that manipulate the files themselves, rather than their data content.

An image stream is an output stream to a display device, such as the display screen or a printer. In addition to the standard output operations, such as print, an image stream implements a variety of graphics operations, such as drawing lines and displaying characters in multiple fonts. Unlike a file, the

"content" of an image stream cannot be retrieved. Image streams are described on page 27.8.

The creation of other kinds of streams, such as network byte-stream connections, is described in the chapters peculiar to those kinds of streams. The operations common to streams in general are described on page 25.1. This chapter describes operations specific to file devices: how to name files, how to open streams to files, and how to manipulate files on their devices.

24.1 Opening and Closing File Streams

In order to perform input from or output to a file, it is necessary to create a stream to the file, using **OPENSTREAM**:

(OPENSTREAM *FILE ACCESS RECOG PARAMETERS* **—)** [Function]

Opens and returns a stream for the file specified by *FILE*, a file name. *FILE* can be either a string or a litatom. The syntax and manipulation of file names is described at length on page 24.5. Incomplete file names are interpreted with respect to the connected directory (page 24.10).

RECOG specifies the recognition mode of *FILE*, as described on page 24.12. If *RECOG* = **NIL**, it defaults according to the value of *ACCESS*.

ACCESS specifies the "access rights" to be used when opening the file, one of the following:

INPUT Only input operations are permitted on the file. The file must already exist. Starts reading at the beginning of the file. *RECOG* defaults to **OLD**.

OUTPUT Only output operations are permitted on the file. Starts writing at the beginning of the file, which is initially empty. While the file is open, other users or processes are unable to open the file for either input or output. *RECOG* defaults to **NEW**.

BOTH Both input and output operations are permitted on the file. Starts reading or writing at the beginning of the file. *RECOG* defaults to **OLD/NEW**. *ACCESS* = **BOTH** implies random accessibility (page 25.18), and thus may not be possible for files on some devices.

APPEND Only sequential output operations are permitted on the file. Starts writing at the *end* of the file. *RECOG* defaults to **OLD/NEW**. *ACCESS* = **APPEND** may not be allowed for files on some devices.

Note: *ACCESS* = **OUTPUT** implies that one intends to write a new or different file, even if a version number was specified and the corresponding file already exists. Thus any previous contents of the file are discarded, and the file is empty immediately after the **OPENSTREAM**. If it is desired to write on an already existing file while preserving the old contents, the file must be opened for access **BOTH** or **APPEND**.

PARAMETERS is a list of pairs (*ATTRIB VALUE*), where *ATTRIB* is any file attribute that the file system is willing to allow the user to set (see **SETFILEINFO**, page 24.17). A non-list *ATTRIB* in *PARAMETERS* is treated as the pair (*ATTRIB* **T**). Generally speaking, attributes that belong to the permanent file (e.g., **TYPE**) can only be set when creating a new file, while attributes that belong only to a particular opening of a file (e.g., **ENDOFSTREAMOP**) can be set on any call to **OPENSTREAM**. Not all devices honor all attributes; those not recognized by a particular device are simply ignored.

In addition to the attributes permitted by **SETFILEINFO**, the following tokens are accepted by **OPENSTREAM** as values of *ATTRIB* in its *PARAMETERS* argument:

DON'T.CHANGE.DATE If *VALUE* is non-**NIL**, the file's creation date (page 24.17) is not changed when the file is opened. This option is meaningful only for old files being opened for access **BOTH**. This should be used only for specialized applications in which the caller does not want the file system to believe the file's content has been changed.

SEQUENTIAL If *VALUE* is non-**NIL**, this opening of the file need support only sequential access; i.e., the caller intends never to use **SETFILEPTR**. For some devices, sequential access to files is much more efficient than random access. Note that the device may choose to ignore this attribute and still open the file in a manner that permits random access. Also note that this attribute does not make sense with *ACCESS* = **BOTH**.

If *FILE* is not recognized by the file system, **OPENSTREAM** causes the error **FILE NOT FOUND**. Ordinarily, this error is intercepted via an entry on **ERRORTYPELST** (page 14.22), which causes **SPELLFILE** (page 24.32) to be called. **SPELLFILE** searches alternate directories and possibly attempts spelling correction on the file name. Only if **SPELLFILE** is unsuccessful will the **FILE NOT FOUND** error actually occur.

If *FILE* exists but cannot be opened, **OPENSTREAM** causes one of several other errors: **FILE WON'T OPEN** if the file is already opened for conflicting access by someone else; **PROTECTION VIOLATION** if the file is protected against the operation; **FILE SYSTEM RESOURCES EXCEEDED** if there is no more room in the file system.

(CLOSEF *FILE***)** [Function]

Closes *FILE*, and returns its full file name. Generates an error, **FILE NOT OPEN**, if *FILE* does not designate an open stream. After closing a stream, no further input/output operations are permitted on it.

If *FILE* is **NIL**, it is defaulted to the primary input stream if that is not the terminal stream, or else the primary output stream if that is not the terminal stream. If both primary input and output streams are the terminal input/output streams, **CLOSEF** returns **NIL**. If **CLOSEF** closes either the primary input stream or the primary output stream (either explicitly or in the *FILE* = **NIL** case), it resets the primary stream for that direction to be the corresponding terminal stream. See page 25.3 for information on the primary input/output streams.

WHENCLOSE (page 24.20) allows the user to "advise" **CLOSEF** to perform various operations when a file is closed.

Because of buffering, the contents of a file open for output are not guaranteed to be written to the actual physical file device until **CLOSEF** is called. Buffered data can be forced out to a file without closing the file by using the function **FORCEOUTPUT** (page 25.10).

Some network file devices perform their transactions in the background. As a result, it is possible for a file to be closed by **CLOSEF** and yet not be "fully" closed for some small period of time afterward, during which time the file appears to still be busy, and cannot be opened for conflicting access by other users.

(CLOSEF? *FILE***)** [Function]

Closes *FILE* if it is open, returning the value of **CLOSEF**; otherwise does nothing and returns **NIL**.

In the present implementation of Interlisp-D, all streams to files are kept, while open, in a registry of "open files". This registry does not include nameless streams, such as string streams (page 24.28), display streams (page 28.29), and the terminal input and output streams; nor streams explicitly hidden from the user, such as dribble streams (page 30.12). This registry may not persist in future implementations of Interlisp-D, but at the present time it is accessible by the following two functions:

(OPENP *FILE ACCESS***)** [Function]

ACCESS is an access mode for a stream opening (one of **INPUT**, **OUTPUT**, **BOTH**, or **APPEND**), or **NIL**, meaning any access.

If *FILE* is a stream, returns its full name if it is open for the specified access, else **NIL**.

If *FILE* is a file name (a litatom), *FILE* is processed according to the rules of file recognition (page 24.12). If a stream open to a file by that name is registered and open for the specified access, then the file's full name is returned. If the file name is not recognized, or no stream is open to the file with the specified access, **NIL** is returned.

If *FILE* is **NIL**, returns a list of the full names of all registered streams that are open for the specified access.

(CLOSEALL *ALLFLG***)** [Function]

Closes all streams in the value of **(OPENP)**. Returns a list of the files closed.

WHENCLOSE (page 24.20) allows certain files to be "protected" from **CLOSEALL**. If *ALLFLG* is **T**, all files, including those protected by **WHENCLOSE**, are closed.

24.2 File Names

A file name in Interlisp-D is a string or litatom whose characters specify a "path" to the actual file: on what host or device the file resides, in which directory, and so forth. Because Interlisp-D supports a variety of non-local file devices, parts of the path could be very device-dependent. However, it is desirable for programs to be able to manipulate file names in a device-independent manner. To this end, Interlisp-D specifies a uniform file name syntax over all devices; the functions that perform the actual file manipulation for a particular device are responsible for any translation to that device's naming conventions.

A file name is composed of a collection of *fields*, some of which have specific semantic interpretations. The functions described below refer to each field by a *field name*, a literal atom from among the following: **HOST**, **DEVICE**, **DIRECTORY**, **NAME**, **EXTENSION**, and **VERSION**. The standard syntax for a file name that contains all of those fields is **{HOST}DEVICE:<DIRECTORY>NAME.EXTENSION;VERSION**. Some host's file systems do not use all of those fields in their file names.

HOST Specifies the host whose file system contains the file. In the case of local file devices, the "host" is the name of the device, e.g., **DSK** or **FLOPPY**.

DEVICE Specifies, for those hosts that divide their file system's name space among mutiple physical devices, the device or logical structure on which the file resides. This should not be confused

with Interlisp-D's abstract "file device", which denotes either a host or a local physical device and is specified by the **HOST** field.

DIRECTORY Specifies the "directory" containing the file. A directory usually is a grouping of a possibly large set of loosely related files, e.g., the personal files of a particular user, or the files belonging to some project. The **DIRECTORY** field usually consists of a principal directory and zero or more subdirectories that together describe a path through a file system's hierarchy. Each subdirectory name is set off from the previous directory or subdirectory by the character "**>**"; e.g., "**LISP>LIBRARY>NEW**".

NAME This field carries no specific meaning, but generally names a set of files thought of as being different renditions of the "same" abstract file.

EXTENSION This field also carries no specific meaning, but generally distinguishes the form of files having the same name. Most files systems have some "conventional" extensions that denote something about the content of the file. E.g., in Interlisp-D, the extension **DCOM** standardly denotes a file containing compiled function definitions.

VERSION A number used to distinguish the versions or "generations" of the files having a common name and extension. The version number is incremented each time a new file by the same name is created.

Most functions that take as input "a directory" accept either a directory name (the contents of the **DIRECTORY** field of a file name) or a "full" directory specification—a file name fragment consisting of only the fields **HOST**, **DEVICE**, and **DIRECTORY**. In particular, the "connected directory" (page 24.10) consists, in general, of all three fields.

For convenience in dealing with certain operating systems, Interlisp-D also recognizes **[]** and **()** as host delimiters (synonymous with **{}**), and **/** as a directory delimiter (synonymous with **<** at the beginning of a directory specification and **>** to terminate directory or subdirectory specification). For example, a file on a Unix file server **UNX** with the name **/usr/foo/bar/stuff.tedit**, whose **DIRECTORY** field is thus **usr/foo/bar**, could be specified as **{UNX}/usr/foo/bar/stuff.tedit**, or **(UNX)<usr/foo/bar>stuff.tedit**, or several other variations. Note that when using **[]** or **()** as host delimiters, they usually must be escaped with the reader's **%** escape character if the file name is expressed as a litatom rather than a string.

Different hosts have different requirements regarding which characters are valid in file names. From Interlisp-D's point of view, any characters are valid. However, in order to be able to parse a file name into its component fields, it is necessary that those characters that are conventionally used as file name delimiters be quoted when they appear inside of fields where

there could be ambiguity. The file name quoting character is "'" (single quote). Thus, the following characters must be quoted when not used as delimeters: :, >, ;, /, and ' itself. The character . (period) need only be quoted if it is to be considered a part of the **EXTENSION** field. The characters },], and) need only be quoted in a file name when the host field of the name is introduced by {, [, and (, respectively. The characters {, [, (, and < need only be quoted if they appear as the first character of a file name fragment, where they would otherwise be assumed to introduce the **HOST** or **DIRECTORY** fields.

The following functions are the standard way to manipulate file names in Interlisp. Their operation is purely syntactic—they perform no file system operations themselves.

(UNPACKFILENAME.STRING *FILENAME* — — —**)** [Function]

Parses *FILENAME*, returning a list in property list format of alternating field names and field contents. The field contents are returned as strings. If *FILENAME* is a stream, its full name is used.

Only those fields actually present in *FILENAME* are returned. A field is considered present if its delimiting punctuation (in the case of **EXTENSION** and **VERSION**, the preceding period or semicolon, respectively) is present, even if the field itself is empty. Empty fields are denoted by "" (the empty string).

Examples:

```
(UNPACKFILENAME.STRING "FOO.BAR") =>
  (NAME "FOO" EXTENSION "BAR")

(UNPACKFILENAME.STRING "FOO.;2") =>
  (NAME "FOO" EXTENSION "" VERSION "2")

(UNPACKFILENAME.STRING "FOO;") =>
  (NAME "FOO" VERSION "")

(UNPACKFILENAME.STRING
  "{ERIS}<LISP>CURRENT>IMTRAN.DCOM;21")
 => (HOST "ERIS" DIRECTORY "LISP>CURRENT"
    NAME "IMTRAN" EXTENSION "DCOM"
    VERSION "21")
```

(UNPACKFILENAME *FILE* —**)** [Function]

Old version of **UNPACKFILENAME.STRING** that returns the field values as atoms, rather than as strings. **UNPACKFILENAME.STRING** is now considered the "correct" way of unpacking file names, because it does not lose information when the contents of a field are numeric. For example,

```
(UNPACKFILENAME 'STUFF.TXT) =>
  (NAME STUFF EXTENSION TXT)
```

but

(UNPACKFILENAME 'STUFF.029) = >
(NAME STUFF EXTENSION 29)

Explicitly omitted fields are denoted by the atom **NIL**, rather than the empty string.

Note: Both **UNPACKFILENAME** and **UNPACKFILENAME.STRING** leave the trailing colon on the device field, so that the Tenex device **NIL:** can be distinguished from the absence of a device. Although **UNPACKFILENAME.STRING** is capable of making the distinction, it retains this behavior for backward compatibility. Thus,

(UNPACKFILENAME.STRING '{TOAST}DSK:FOO) = >
(HOST "TOAST" DEVICE "DSK:" NAME "FOO")

(FILENAMEFIELD *FILENAME FIELDNAME***)** [Function]

Returns, as an atom, the contents of the *FIELDNAME* field of *FILENAME*. If *FILENAME* is a stream, its full name is used.

(PACKFILENAME.STRING $FIELD_1$ $CONTENTS_1$ ... $FIELD_N$ $CONTENTS_N$**)** [NoSpread Function]

Takes a sequence of alternating field names and field contents (atoms or strings), and returns the corresponding file name, as a string.

If **PACKFILENAME.STRING** is given a single argument, it is interpreted as a list of alternating field names and field contents. Thus **PACKFILENAME.STRING** and **UNPACKFILENAME.STRING** operate as inverses.

If the same field name is given twice, the *first* occurrence is used.

The contents of the field name **DIRECTORY** may be either a directory name or a full directory specification as described above.

PACKFILENAME.STRING also accepts the "field name" **BODY** to mean that its contents should itself be unpacked and spliced into the argument list at that point. This feature, in conjunction with the rule that fields early in the argument list override later duplicates, is useful for altering existing file names. For example, to provide a default field, place **BODY** first in the argument list, then the default fields. To override a field, place the new fields first and **BODY** last.

If the value of the **BODY** field is a stream, its full name is used.

Examples:

(PACKFILENAME.STRING 'DIRECTORY "LISP"
'NAME "NET")
= > "<LISP>NET"

```
(PACKFILENAME.STRING 'NAME "NET"
  'DIRECTORY "{DSK}<LISPFILES>")
    => "{DSK}<LISPFILES>NET"

(PACKFILENAME.STRING 'DIRECTORY "{DSK}"
  'BODY "{TOAST}<FOO>BAR")
    => "{DSK}BAR"

(PACKFILENAME.STRING 'DIRECTORY "FRED"
  'BODY "{TOAST}<FOO>BAR")
    => "{TOAST}<FRED>BAR"

(PACKFILENAME.STRING 'BODY "{TOAST}<FOO>BAR"
  'DIRECTORY "FRED")
    => "{TOAST}<FOO>BAR"

(PACKFILENAME.STRING 'VERSION NIL
  'BODY "{TOAST}<FOO>BAR.DCOM;2")
    => "{TOAST}<FOO>BAR.DCOM"

(PACKFILENAME.STRING 'BODY "{TOAST}<FOO>BAR.DCOM"
  'VERSION 1)
    => "{TOAST}<FOO>BAR.DCOM;1"

(PACKFILENAME.STRING 'BODY "{TOAST}<FOO>BAR.DCOM;"
  'VERSION 1)
    => "{TOAST}<FOO>BAR.DCOM;"

(PACKFILENAME.STRING 'BODY "BAR.;1"
  'EXTENSION "DCOM")
    => "BAR.;1"

(PACKFILENAME.STRING 'BODY "BAR;1"
  'EXTENSION "DCOM")
    => "BAR.DCOM;1"
```

In the last two examples, note that in one case the extension is explicitly present in the body (as indicated by the preceding period), while in the other there is no indication of an extension, so the default is used.

(PACKFILENAME *FIELD*$_1$ *CONTENTS*$_1$ ... *FIELD*$_N$ *CONTENTS*$_N$**)** [NoSpread Function]

The same as **PACKFILENAME.STRING**, except that it returns the file name as a litatom, instead of a string.

24.3 Incomplete File Names

In general, it is not necessary to pass a complete file name (one containing all the fields listed above) to functions that take a file name as argument. Interlisp supplies suitable defaults for

certain fields, as described below. Functions that return names of actual files, however, always return the fully specified name.

If the version field is omitted from a file name, Interlisp performs version recognition, as described on page 24.11.

If the host, device and/or directory field are omitted from a file name, Interlisp defaults them with respect to the currently connected directory. The connected directory is changed by calling the function **CNDIR** or using the programmer's assistant command **CONN**.

Defaults are added to the partially specified name "left to right" until a host, device or directory field is encountered. Thus, if the connected directory is **{TWENTY}PS:<FRED>**, then

BAR.DCOM means
{TWENTY}PS:<FRED>BAR.DCOM

<GRANOLA>BAR.DCOM means
{TWENTY}PS:<GRANOLA>BAR.DCOM

MTA0:<GRANOLA>BAR.DCOM means
{TWENTY}MTA0:<GRANOLA>BAR.DCOM

{THIRTY}<GRANOLA>BAR.DCOM means
{THIRTY}<GRANOLA>BAR.DCOM

In addition, if the partially specified name contains a subdirectory, but no principal directory, then the subdirectory is appended to the connected directory. For example,

ISO>BAR.DCOM means
{TWENTY}PS:<FRED>ISO>BAR.DCOM

Or, if the connected directory is the Unix directory **{UNX}/usr/fred/**, then **iso/bar.dcom** means **{UNX}/usr/fred/iso/bar.dcom**, but **/other/bar.dcom** means **{UNX}/other/bar.dcom**.

(CNDIR *HOST/DIR***)** [Function]

Connects to the directory *HOST/DIR*, which can either be a directory name or a full directory specification including host and/or device. If the specification includes just a host, and the host supports directories, the directory is defaulted to the value of **(USERNAME)**; if the host is omitted, connection is made to another directory on the same host as before. If *HOST/DIR* is **NIL**, connects to the value of **LOGINHOST/DIR**.

CNDIR returns the full name of the now-connected directory. Causes an error, **Non-existent directory**, if *HOST/DIR* is not recognized as a valid directory.

Note that **CNDIR** does not necessarily require or provide any directory access privileges. Access privileges are checked when a file is opened.

CONN *HOST/DIR* [Prog. Asst. Command]

Convenient command form of **CNDIR** for use at the executive. Connects to *HOST/DIR*, or to the value of **LOGINHOST/DIR** if *HOST/DIR* is omitted. This command is undoable—undoing it causes the system to connect to the previously connected directory.

LOGINHOST/DIR [Variable]

CONN with no argument connects to the value of the variable **LOGINHOST/DIR**, initially **{DSK}**, but usually reset in the user's greeting file (page 12.1).

(DIRECTORYNAME *DIRNAME STRPTR* —**)** [Function]

If *DIRNAME* is **T**, returns the full specification of the currently connected directory. If *DIRNAME* is **NIL**, returns the "login" directory specification (the value of **LOGINHOST/DIR**). For any other value of *DIRNAME*, returns a full directory specification if *DIRNAME* designates an existing directory (satisfies **DIRECTORYNAMEP**), otherwise **NIL**.

If *STRPTR* is **T**, the value is returned as an atom, otherwise it is returned as a string.

(DIRECTORYNAMEP *DIRNAME HOSTNAME***)** [Function]

Returns **T** if *DIRNAME* is recognized as a valid directory on host *HOSTNAME*, or on the host of the currently connected directory if *HOSTNAME* is **NIL**. *DIRNAME* may be either a directory name or a full directory specification containing host and/or device as well.

If *DIRNAME* includes subdirectories, this function may or may not pass judgment on their validity. Some hosts support "true" subdirectories, distinct entities manipulable by the file system, while others only provide them as a syntactic convenience.

(HOSTNAMEP *NAME***)** [Function]

Returns **T** if *NAME* is recognized as a valid host or file device name at the moment **HOSTNAMEP** is called.

24.4 Version Recognition

Most of the file devices in Interlisp support file version numbers. That is, it is possible to have several files of the exact same name, differing only in their **VERSION** field, which is incremented for each new "version" of the file that is created. When a file name lacking a version number is presented to the file system, it is

necessary to determine which version number is intended. This process is known as *version recognition*.

When **OPENSTREAM** opens a file for input and no version number is given, the highest existing version number is used. Similarly, when a file is opened for output and no version number is given, a new file is created with a version number one higher than the highest one currently in use with that file name. The version number defaulting for **OPENSTREAM** can be changed by specifying a different value for its *RECOG* argument, as described under **FULLNAME**, below.

Other functions that accept file names as arguments generally perform the default version recognition, which is newest version for existing files, or a new version if using the file name to create a new file. The one exception is **DELFILE**, which defaults to the oldest existing version of the file.

The functions below can be used to perform version recognition without actually calling **OPENSTREAM** to open the file. Note that these functions only tell the truth about the moment at which they are called, and thus cannot in general be used to anticipate the name of the file opened by a comparable **OPENSTREAM**. They are sometimes, however, helpful hints.

(FULLNAME *X RECOG***)** [Function]

If *X* is an open stream, simply returns the full file name of the stream. Otherwise, if *X* is a file name given as a string or litatom, performs version recognition, as follows:

If *X* is recognized in the recognition mode specified by *RECOG* as an abbreviation for some file, returns the file's full name, otherwise **NIL**. *RECOG* is one of the following:

OLD Choose the newest existing version of the file. Return **NIL** if no file named *X* exists.

OLDEST Choose the oldest existing version of the file. Return **NIL** if no file named *X* exists.

NEW Choose a new (not yet existing) version of the file. That is, if versions of *X* already exist, then choose a version number one higher than highest existing version; else choose version 1. For some file systems, **FULLNAME** returns **NIL** if the user does not have the access rights necessary for creating a new file named *X*.

OLD/NEW Try **OLD**, then **NEW**. That is, choose the newest existing version of the file, if any; else choose version 1. This usually only makes sense if you are intending to open *X* for access **BOTH**.

RECOG = **NIL** defaults to **OLD**. For all other values of *RECOG*, generates an error **ILLEGAL ARG**.

If *X* already contains a version number, the *RECOG* argument will never change it. In particular, *RECOG* = **NEW** does not require

that the file actually be new. For example, **(FULLNAME 'FOO.;2 'NEW)** may return **{ERIS}<LISP>FOO.;2** if that file already exists, even though **(FULLNAME 'FOO 'NEW)** would default the version to a new number, perhaps returning **{ERIS}<LISP>FOO.;5**.

(INFILEP *FILE***)** [Function]

Equivalent to **(FULLNAME** *FILE* **'OLD)**. That is, returns the full file name of the newest version of *FILE* if *FILE* is recognized as specifying the name of an existing file that could potentially be opened for input, **NIL** otherwise.

(OUTFILEP *FILE***)** [Function]

Equivalent to **(FULLNAME** *FILE* **'NEW)**.

Note that **INFILEP**, **OUTFILEP** and **FULLNAME** do not open any files; they are pure predicates. In general they are also only hints, as they do not necessarily imply that the caller has access rights to the file. For example, **INFILEP** might return non-**NIL**, but **OPENSTREAM** might fail for the same file because the file is read-protected against the user, or the file happens to be open for output by another user at the time. Similarly, **OUTFILEP** could return non-**NIL**, but **OPENSTREAM** could fail with a **FILE SYSTEM RESOURCES EXCEEDED** error.

Note also that in a shared file system, such as a remote file server, intervening file operations by another user could contradict the information returned by recognition. For example, a file that was **INFILEP** might be deleted, or between an **OUTFILEP** and the subsequent **OPENSTREAM**, another user might create a new version or delete the highest version, causing **OPENSTREAM** to open a different version of the file than the one returned by **OUTFILEP**. In addition, some file servers do not well support recognition of files in output context. Thus, in general, the "truth" about a file can only be obtained by actually opening the file; creators of files should rely on the name of the stream opened by **OPENSTREAM**, not the value returned from these recognition functions. In particular, for the reasons described earlier, programmers are discouraged from using **OUTFILEP** or **(FULLNAME** *NAME* **'NEW)**.

24.5 Using File Names Instead of Streams

In earlier implementations of Interlisp, from the days of Interlisp-10 onward, the "handle" used to refer to an open file was not a stream, but rather the file's full name, represented as a

litatom. When the file name was passed to any I/O function, it was mapped to a stream by looking it up in a list of open files. This scheme was sometimes convenient for typing in file commands at the executive, but was very poor for serious programming in two major ways. First, the mapping from file name to stream on every input/output operation is inefficient. Second, and more importantly, using the file name as the handle on an open stream means that it is not possible to have more than one stream open on a given file at once.

As of this writing, Interlisp-D is in a transition period, where it still supports the use of litatom file names as synonymous with open streams, but this use is not recommended. The remainder of this section discusses this usage of file names for the benefit of those reading older programs and wishing to convert them as necessary to work properly when this compatibility feature is removed.

24.5.1 File Name Efficiency Considerations

It is possible for a program to be seriously inefficient using a file name as a stream if the program is not using the file's full name, the name returned by **OPENFILE** (below). Any time that an input/output function is called with a file name other than the full file name, Interlisp must perform recognition on the partial file name in order to determine which open file is intended. Thus if repeated operations are to be performed, it is considerably more efficient to use the full file name returned from **OPENFILE** than to repeatedly use the possibly incomplete name that was used to open the file.

There is a more subtle problem with partial file names, in that recognition is performed on the user's entire directory, not just the open files. It is possible for a file name that was previously recognized to denote one file to suddenly denote a different file. For example, suppose a program performs **(INFILE 'FOO)**, opening **FOO.;1**, and reads several expressions from **FOO**. Then the user interrupts the program, creates a **FOO.;2** and resumes the program (or a user at another workstation creates a **FOO.;2**). Now a call to **READ** giving it **FOO** as its *FILE* argument will generate a **FILE NOT OPEN** error, because **FOO** will be recognized as **FOO.;2**.

24.5.2 Obsolete File Opening Functions

The following functions are now considered obsolete, but are provided for backwards compatibility:

(OPENFILE *FILE ACCESS RECOG PARAMETERS* **—)** [Function]

Opens *FILE* with access rights as specified by *ACCESS*, and recognition mode *RECOG*, and returns the full name of the resulting stream. Equivalent to **(FULLNAME (OPENSTREAM** *FILE ACCESS RECOG PARAMETERS***))**.

(INFILE *FILE***)** [Function]

Opens *FILE* for input, and sets it as the primary input stream. Equivalent to **(INPUT (OPENSTREAM** *FILE* **'INPUT 'OLD))**

(OUTFILE *FILE***)** [Function]

Opens *FILE* for output, and sets it as the primary output stream. Equivalent to **(OUTPUT (OPENSTREAM** *FILE* **'OUTPUT 'NEW))**.

(IOFILE *FILE***)** [Function]

Equivalent to **(OPENFILE** *FILE* **'BOTH 'OLD)**; opens *FILE* for both input and output. Does not affect the primary input or output stream.

24.5.3 Converting Old Programs

At some point in the future, the Interlisp-D file system will change so that each call to **OPENSTREAM** returns a distinct stream, even if a stream is already open to the specified file. This change is required in order to deal rationally with files in a multiprocessing environment.

This change will of necessity produce the following incompatibilities:

1) The functions **OPENFILE**, **INPUT**, and **OUTPUT** will return a **STREAM**, not a full file name. To make this less confusing in interactive situations, **STREAM**s will have a print format that reveals the underlying file's actual name,

2) A greater penalty will ensue for passing as the *FILE* argument to i/o operations anything other than the object returned from **OPENFILE**. Passing the file's name will be significantly slower than passing the stream (even when passing the "full" file name), and in the case where there is more than one stream open on the file it might even act on the wrong one.

3) **OPENP** will return **NIL** when passed the name of a file rather than a stream (the value of **OPENFILE** or **OPENSTREAM**).

Users should consider the following advice when writing new programs and editing existing programs, in order that they will continue to operate well when this change is made:

Because of the efficiency and ambiguity considerations described earlier, users have long been encouraged to use only full file

names as *FILE* arguments to i/o operations. The "proper" way to have done this was to bind a variable to the value returned from **OPENFILE** and pass that variable to all i/o operations; such code will continue to work. A less proper way to obtain the full file name, but one which has to date not incurred any obvious penalty, is that which binds a variable to the result of an **INFILEP** and passes that to **OPENFILE** and all i/o operations. This has worked because **INFILEP** and **OPENFILE** both return a full file name, an invalid assumption in this future world. Such code should be changed to pass around the value of the **OPENFILE**, not the **INFILEP**.

Code that calls **OPENP** to test whether a possibly incomplete file name is already open should be recoded to pass to **OPENP** only the value returned from **OPENFILE** or **OPENSTREAM**.

Code that uses ordinary string functions to manipulate file names, and in particular the value returned from **OPENFILE**, should be changed to use the the functions **UNPACKFILENAME.STRING** and **PACKFILENAME.STRING**. Those functions work both on file names (strings) and streams (coercing the stream to the name of its file).

Code that tests the value of **OUTPUT** for equality to some known file name or **T** should be examined carefully and, if possible, recoded.

To see more directly the effects of passing around **STREAM**s instead of file names, replace your calls to **OPENFILE** with calls to **OPENSTREAM**. **OPENSTREAM** is called in exactly the same way, but returns a **STREAM**. Streams can be passed to **READ**, **PRINT**, **CLOSEF**, etc just as the file's full name can be currently, but using them is more efficient. The function **FULLNAME**, when applied to a stream, returns its full file name.

24.6 Using Files with Processes

Because Interlisp-D does not yet support multiple streams per file, problems can arise if different processes attempt to access the same file. The user has to be careful not to have two processes manipulating the same file at the same time, since the two processes will be sharing a single input stream and file pointer. For example, it will not work to have one process **TCOMPL** a file while another process is running **LISTFILES** on it.

24.7 File Attributes

Any file has a number of "file attributes", such as the read date, protection, and bytesize. The exact attributes that a file can have is dependent on the file device. The functions **GETFILEINFO** and **SETFILEINFO** allow the user to conveniently access file attributes:

(GETFILEINFO *FILE ATTRIB***)** [Function]

Returns the current setting of the *ATTRIB* attribute of *FILE*.

(SETFILEINFO *FILE ATTRIB VALUE***)** [Function]

Sets the attribute *ATTRIB* of *FILE* to be *VALUE*. **SETFILEINFO** returns **T** if it is able to change the attribute *ATTRIB*, and **NIL** if unsuccessful, either because the file device does not recognize *ATTRIB* or because the file device does not permit the attribute to be modified.

The *FILE* argument to **GETFILEINFO** and **SETFILEINFO** can be an open stream (or an argument designating an open stream, see page 25.2), or the name of a closed file. **SETFILEINFO** in general requires write access to the file.

The attributes recognized by **GETFILEINFO** and **SETFILEINFO** fall into two categories: *permanent* attributes, which are properties of the file, and *temporary* attributes, which are properties only of an open stream to the file. The temporary attributes are only recognized when *FILE* designates an open stream; the permanent attributes are usually equally accessible for open and closed files. However, some devices are willing to change the value of certain attributes of an open stream only when specified in the *PARAMETERS* argument to **OPENSTREAM** (page 24.2), not on a later call to **SETFILEINFO**.

The following are currently recognized as permanent attributes of a file:

BYTESIZE The byte size of the file. Interlisp-D currently only supports byte size 8.

LENGTH The number of bytes in the file. Alternatively, the byte position of the end-of-file. Like **(GETEOFPTR** *FILE***)**, but *FILE* does not have to be open.

SIZE The size of *FILE* in pages.

CREATIONDATE The date and time, as a string, that the content of *FILE* was "created". The creation date changes whenever the content of the file is modified, but remains unchanged when a file is transported, unmodified, across file systems. Specifically, **COPYFILE** and **RENAMEFILE** (page 24.31) preserve the file's creation date. Note that this is different from the concept of "creation date" used by some operating systems (e.g., Tops20).

WRITEDATE The date and time, as a string, that the content of *FILE* was last written to this particular file system. When a file is copied, its creation date does not change, but its write date becomes the time at which the copy is made.

READDATE The date and time, as a string, that *FILE* was last read, or **NIL** if it has never been read.

ICREATIONDATE
IWRITEDATE
IREADDATE The **CREATIONDATE**, **WRITEDATE** and **READDATE**, respectively, in integer form, as **IDATE** (page 12.14) would return. This form is useful for comparing dates.

AUTHOR The name of the user who last wrote the file.

TYPE The "type" of the file, some indication of the nature of the file's content. The "types" of files allowed depends on the file device. Most devices recognize the litatom **TEXT** to mean that the file contains just characters, or **BINARY** to mean that the file contains arbitrary data.

Some devices support a wider range of file types that distinguish among the various sorts of files one might create whose content is "binary". All devices interpret any value of **TYPE** that they do not support to be **BINARY**. Thus, **GETFILEINFO** may return the more general value **BINARY** instead of the original type that was passed to **SETFILEINFO** or **OPENSTREAM**. Similarly, **COPYFILE**, while attempting to preserve the **TYPE** of the file it is copying, may turn, say, an **INTERPRESS** file into a mere **BINARY** file.

The way in which some file devices (e.g., Xerox file servers) support a wide range of file types is by representing the type as an integer, whose interpretation is known by the client. The variable **FILING.TYPES** is used to associate symbolic types with numbers for these devices. This list initially contains some of the well-known assignments of type name to number; the user can add additional elements to handle any private file types. For example, suppose there existed an NS file type **MAZEFILE** with numeric value 5678. You could add the element **(MAZEFILE 5678)** to **FILING.TYPES** and then use **MAZEFILE** as a value for the **TYPE** attribute to **SETFILEINFO** or **OPENSTREAM**. Other devices are, of course, free to store **TYPE** attributes in whatever manner they wish, be it numeric or symbolic. **FILING.TYPES** is merely considered the official registry for Xerox file types.

For most file devices, the **TYPE** of a newly created file, if not specified in the *PARAMETERS* argument to **OPENSTREAM**, defaults to the value of **DEFAULTFILETYPE**, initially **TEXT**.

The following are currently recognized as temporary attributes of an open stream:

ACCESS The current access rights of the stream (see page 24.2). Can be one of **INPUT**, **OUTPUT**, **BOTH**, **APPEND**; or **NIL** if the stream is not open.

ENDOFSTREAMOP The action to be taken when a stream is at "end of file" and an attempt is made to take input from it. The value of this attribute is a function of one argument, the stream. The function can examine the stream and its calling context and take any action it wishes. If the function returns normally, its should return either **T**, meaning to try the input operation again, or the byte that **BIN** would have returned had there been more bytes to read. Ordinarily, one should not let the **ENDOFSTREAMOP** function return unless one is only performing binary input from the file, since there is no way in general of knowing in what state the reader was at the time the end of file occurred, and hence how it will interpret a single byte returned to it.

The default **ENDOFSTREAMOP** is a system function that causes the error **END OF FILE**. The behavior of that error can be further modified for a particular stream by using the **EOF** option of **WHENCLOSE** (page 24.20).

EOL The end-of-line convention for the stream. This can be **CR**, **LF**, or **CRLF**, indicating with what byte or sequence of bytes the "End Of Line" character is represented on the stream. On input, that sequence of bytes on the stream is read as **(CHARCODE EOL)** by **READCCODE** or the string reader. On output, **(TERPRI)** and **(PRINTCCODE (CHARCODE EOL))** cause that sequence of bytes to be placed on the stream.

The end of line convention is usually not apparent to the user. The file system is usually aware of the convention used by a particular remote operating system, and sets this attribute accordingly. If you believe a file actually is stored with a different convention than the default, it is possible to modify the default behavior by including the **EOL** attribute in the *PARAMETERS* argument to **OPENSTREAM**.

BUFFERS Value is the number of 512-byte buffers that the stream maintains at one time. This attribute is only used by certain random-access devices (currently, the local disk, floppy, and Leaf servers); all others ignore it.

Streams open to files generally maintain some portion of the file buffered in memory, so that each call to an I/O function does not require accessing the actual file on disk or a file server. For files being read or written sequentially, not much buffer space is needed, since once a byte is read or written, it will never need to be seen again. In the case of random access streams, buffering is more complicated, since a program may jump around in the file, using **SETFILEPTR** (page 25.19). In this case, the more buffer space the stream has, the more likely it is that after a **SETFILEPTR** to a place in the file that has already been accessed, the stream

still has that part of the file buffered and need not go out to the device again. This benefit must, of course, be traded off against the amount of memory consumed by the buffers.

24.8 Closing and Reopening Files

The function **WHENCLOSE** permits the user to associate certain operations with open streams that govern how and when the stream will be closed. The user can specify that certain functions will be executed before **CLOSEF** closes the stream and/or after **CLOSEF** closes the stream. The user can make a particular stream be invisible to **CLOSEALL**, so that it will remain open across user invocations of **CLOSEALL**.

(WHENCLOSE *FILE PROP*$_1$ *VAL*$_1$ ... *PROP*$_N$ *VAL*$_N$**)** [NoSpread Function]

FILE must designate an open stream other than **T** (**NIL** defaults to the primary input stream, if other than **T**, or primary output stream if other than **T**). The remaining arguments specify properties to be associated with the full name of *FILE*. **WHENCLOSE** returns the full name of *FILE* as its value.

WHENCLOSE recognizes the following property names:

BEFORE *VAL* is a function that **CLOSEF** will apply to the stream just before it is closed. This might be used, for example, to copy information about the file from an in-core data structure to the file just before it is closed.

AFTER *VAL* is a function that **CLOSEF** will apply to the stream just after it is closed. This capability permits in-core data structures that know about the stream to be cleaned up when the stream is closed.

CLOSEALL *VAL* is either **YES** or **NO** and determines whether *FILE* will be closed by **CLOSEALL** (**YES**) or whether **CLOSEALL** will ignore it (**NO**). **CLOSEALL** uses **CLOSEF**, so that any **AFTER** functions will be executed if the stream is in fact closed. Files are initialized with **CLOSEALL** set to **YES**.

EOF *VAL* is a function that will be applied to the stream when an end-of-file error occurs, and the **ERRORTYPELST** entry for that error, if any, returns **NIL**. The function can examine the context of the error, and can decide whether to close the stream, **RETFROM** some function, or perform some other computation. If the function supplied returns normally (i.e., does not **RETFROM** some function), the normal error machinery will be invoked.

The default **EOF** behavior, unless overridden by this **WHENCLOSE** option, is to call the value of **DEFAULTEOFCLOSE** (below).

For some applications, the **ENDOFSTREAMOP** attribute (page 24.19) is a more useful way to intercept the end-of-file error. The **ENDOFSTREAMOP** attribute comes into effect before the error machinery is ever activated.

Multiple **AFTER** and **BEFORE** functions may be associated with a file; they are executed in sequence with the most recently associated function executed first. The **CLOSEALL** and **EOF** values, however, will override earlier values, so only the last value specified will have an effect.

DEFAULTEOFCLOSE [Variable]

Value is the name of a function that is called by default when an end of file error occurs and no **EOF** option has been specified for the stream by **WHENCLOSE**. The initial value of **DEFAULTEOFCLOSE** is **NILL**, meaning take no special action (go ahead and cause the error). Setting it to **CLOSEF** would cause the stream to be closed before the rest of the error machinery is invoked.

24.9 Local Hard Disk Device

Warning: This section describes the Interlisp-D functions that control the local hard disk drive available on some computers. All of these functions may not work on all computers running Interlisp-D. For more information on using the local hard disk facilities, see the users guide for your computer.

This section describes the local file system currently supported on the Xerox 1108 and 1186 computers. The Xerox 1132 supports a simpler local file system. The functions below are no-ops on the Xerox 1132, except for **DISKPARTITION** (which returns a disk partition number), and **DISKFREEPAGES**. On the Xerox 1132, different numbered partitions are referenced by using devices such as **{DSK1}**, **{DSK2}**, etc. **{DSK}** always refers to the disk partition that Interlisp is running on. The 1132 local file system does not support the use of directories.

The hard disk used with the Xerox 1108 or 1186 may be partitioned into a number of named "logical volumes." Logical volumes may be used to hold the Interlisp virtual memory file (see page 12.6), or Interlisp files. For information on intializing and partitioning the hard disk, see the users guide for your computer. In order to store Interlisp files on a logical volume, it is necessary to create a lisp file directory on that volume (see **CREATEDSKDIRECTORY**, below).

So long as there exists a logical volume with a Lisp directory on it, files on this volume can be accessed by using the file device called **{DSK}**. Interlisp-D can be used to read, write, and otherwise

interact with files on local disk disks through standard Interlisp input/output functions. All I/O functions such as **LOAD**, **OPENSTREAM**, **READ**, **PRINT**, **GETFILEINFO**, **COPYFILE**, etc., work with files on the local disk.

If you do not have a logical volume with a Lisp directory on it, Interlisp emulates the **{DSK}** device by a core device, a file device whose backing store is entirely within the Lisp virtual memory. However, this is not recommended because the core device only provides limited scratch space, and since the core device is contained in virtual memory, it (and the files stored on it) will be erased when the virtual memory file is reloaded.

Each logical volume with a Lisp directory on it serves as a directory of the device **{DSK}**. Files are referred to by forms such as

{DSK}< *VOLUMENAME* **>** *FILENAME*

Thus, the file **INIT.LISP** on the volume **LISPFILES** would be called **{DSK}<LISPFILES>INIT.LISP**.

Subdirectories within a logical volume are supported, using the > character in file names to delimit subdirectory names. For example, the file name **{DSK}<LISPFILES>DOC>DESIGN.TEDIT** designates the file names **DESIGN.TEDIT** on the subdirectory **DOC** on the logical volume **LISPFILES**.

If a logical volume name is not specified, it defaults in an unusual but simple way: the logical volume defaults to the next logical volume that has a lisp file directory on it including or after the volume containing the currently running virtual memory. For example, if the local disk has the logical volumes **LISP**, **TEMP**, and **LISPFILES**, the **LISP** volume contains the running virtual memory, and only the **LISP** volume has a Lisp file directory on it, then **{DSK}INIT.LISP** refers to the file **{DSK}<LispFiles>INIT.LISP**. All the functions below default logical volume names in a similar way, except for those such as **CREATEDSKDIRECTORY**. To determine the current default lisp file directory, evaluate **(DIRECTORYNAME '{DSK})**.

(CREATEDSKDIRECTORY *VOLUMENAME* **—)** [Function]

Creates a lisp file directory on the logical volume *VOLUMENAME*, and returns the name of the directory created. It is only necessary to create a lisp file directory the first time the logical volume is used. After that, the system automatically recognizes and opens access to the logical volumes that have lisp file directories on them.

(PURGEDSKDIRECTORY *VOLUMENAME* **—)** [Function]

Erases all lisp files on the volume *VOLUMENAME*, and deletes the lisp file directory.

(LISPDIRECTORYP *VOLUMENAME***)** [Function]

Returns **T** if the logical volume *VOLUMENAME* has a lisp file directory on it.

(VOLUMES) [Function]

Returns a list of the names of all of the logical volumes on the local hard disk (whether they have lisp file directories or not).

(VOLUMESIZE *VOLUMENAME* —**)** [Function]

Returns the total size of the logical volume *VOLUMENAME* in disk pages.

(DISKFREEPAGES *VOLUMENAME* —**)** [Function]

Returns the total number of free disk pages left on the logical volume *VOLUMENAME*.

(DISKPARTITION) [Function]

Returns the name of the logical volume containing the virtual memory file that Interlisp is currently running in (see page 12.6).

(DSKDISPLAY *NEWSTATE***)** [Function]

Controls a display window that displays information about the logical volumes on the local hard disk (logical volume names, sizes, free pages, etc.). **DSKDISPLAY** opens or closes this display window depending on the value of *NEWSTATE* (one of **ON**, **OFF**, or **CLOSED**), and returns the previous state of the display window.

If *NEWSTATE* is **ON**, the display window is opened, and it is automatically updated whenever the file system state changes (this can slow file operations significantly). If *NEWSTATE* is **OFF**, the display window is opened, but it is not automatically updated. If *NEWSTATE* is **CLOSED**, the display window is closed. The display mode is initially set to **CLOSED**.

Once the display window is open, the user can update it or change its state with the mouse. Left-buttoning the display window updates it, and middle-buttoning the window brings up a menu that allows you to change the display state.

Note: **DSKDISPLAY** uses the value of the variable **DSKDISPLAY.POSITION** for the position of the lower-left corner of the disk display window when it is opened. This variable is changed if the disk display window is moved.

(SCAVENGEDSKDIRECTORY *VOLUMENAME SILENT***)** [Function]

Rebuilds the lisp file directory for the logical volume *VOLUMENAME*. This may repair damage in the unlikely event of

file system failure, signified by symptoms such as infinite looping or other strange behavior while the system is doing a directory search. Calling **SCAVENGEDSKDIRECTORY** will not harm an intact volume.

Normally, **SCAVENGEDSKDIRECTORY** prints out messages as it scavenges the directory. If *SILENT* is non-**NIL**, these messages are not printed.

Note: Some low-level disk failures may cause "**HARD DISK ERROR**" errors to occur. To fix such a failure, it may be necessary to log out of Interlisp, scavenge the logical volume in question using Pilot tools, and then call **SCAVENGEDSKDIRECTORY** from within Interlisp. See the users guide for your computer for more information.

24.10 Floppy Disk Device

Warning: This section describes the Interlisp-D functions that control the floppy disk drive available on some computers. All of these functions may not work on all computers running Interlisp-D. For more information on using the floppy disk facilities, see the users guide for your computer.

The floppy disk drive is accessed through the device **{FLOPPY}**. Interlisp-D can be used to read, write, and otherwise interact with files on floppy disks through standard Interlisp input/output functions. All I/O functions such as **LOAD**, **OPENSTREAM**, **READ**, **PRINT**, **GETFILEINFO**, **COPYFILE**, etc., work with files on floppies.

Note that floppy disks are a removable storage medium. Therefore, it is only meaningful to perform i/o operations to the floppy disk drive, rather than to a given floppy disk. In this section, the phrase "the floppy" is used to mean "the floppy that is currently in the floppy disk drive."

For example, the following sequence could be used to open a file **XXX.TXT** on the floppy, print "Hello" on it, and close it:

```
(SETQ XXX (OPENSTREAM '{FLOPPY}XXX.TXT 'OUTPUT 'NEW)
(PRINT "Hello" XXX)
(CLOSEF XXX)
```

(FLOPPY.MODE *MODE***)** [Function]

Interlisp-D can currently read and write files on floppies stored in a number of different formats. At any point, the floppy is considered to be in one of four "modes," which determines how it reads and writes files on the floppy. **FLOPPY.MODE** sets the floppy mode to the value of *MODE*, one of **PILOT**, **HUGEPILOT**, **SYSOUT**, or **CPM**, and returns the previous floppy mode. The floppy modes are interpreted as follows:

PILOT This is the normal floppy mode, using floppies in the Xerox Pilot floppy disk format. This file format allows all of the normal Interlisp-D I/O operations. This format also supports file names with arbitrary levels of subdirectories. For example, it is possible to create a file named **{FLOPPY}<Lisp>Project>FOO.TXT**.

HUGEPILOT This floppy mode is used to access files that are larger than a single floppy, stored on multiple floppies. There are some restrictions with using "huge" files. Some I/O operations are not meaningful for "huge" files. When a stream is created for output in this mode, the **LENGTH** file attribute (page 24.17) must be specified when the file is opened, so that it is known how many floppies will be needed. When an output file is created, the floppy (or floppies) are automatically erased and reformatted (after confirmation from the user).

HUGEPILOT mode is primarily useful for saving big files to and from floppies. For example, the following could be used to copy the file **{ERIS}<Lisp>Bigfile.txt** onto the huge Pilot file **{FLOPPY}BigFile.save**:

```
(FLOPPY.MODE 'HUGEPILOT)
(COPYFILE '{ERIS}<Lisp>Bigfile.txt '{FLOPPY}BigFile.save)
```

and the following would restore the file:

```
(FLOPPY.MODE 'HUGEPILOT)
(COPYFILE '{FLOPPY}BigFile.save '{ERIS}<Lisp>Bigfile.txt)
```

During each copying operation, the user will be prompted to insert "the next floppy" if **{ERIS}<Lisp>Bigfile.txt** takes multiple floppies.

SYSOUT Similar to **HUGEPILOT** mode, **SYSOUT** mode is used for storing sysout files (page 12.8) on multiple floppy disks. The user is prompted to insert new floppies as they are needed.

This mode is set automatically when **SYSOUT** or **MAKESYS** is done to the floppy device: **(SYSOUT '{FLOPPY})** or **(MAKESYS '{FLOPPY})**. Notice that the file name does not need to be specifed in **SYSOUT** mode; unlike **HUGEPILOT** mode, the file name **Lisp.sysout** is always used.

Note: The procedure for loading sysout files from floppies depends on the particular computer being used. For information on loading sysout files from floppies, see the users guide for your computer.

Explicitly setting the mode to **SYSOUT** is useful when copying a sysout file to or from floppies. For example, the following can be used to copy the sysout file **{ERIS}<Lisp>Lisp.sysout** onto floppies (it is important to set the floppy mode back when done):

```
(FLOPPY.MODE 'SYSOUT)
(COPYFILE '{ERIS}<Lisp>Lisp.sysout '{FLOPPY})
```

(FLOPPY.MODE 'PILOT)

CPM Interlisp-D supports the single-density single-sided (SDSS) CPM floppy format (a standard used by many computers). CPM-formatted floppies are totally different than Pilot floppies, so the user should call **FLOPPY.MODE** to switch to **CPM** mode when planning to use CPM floppies. After switching to **CPM** mode, **FLOPPY.FORMAT** can be used to create CPM-formatted floppies, and the usual input/output operations work with CPM floppy files.

Note: There are a few limitations on CPM floppy format files: (1) CPM file names are limited to eight or fewer characters, with extensions of three or fewer characters; (2) CPM floppies do not have directories or version numbers; and (3) CPM files are padded out with blanks to make the file lengths multiples of 128.

(FLOPPY.FORMAT *NAME AUTOCONFIRMFLG SLOWFLG***)** [Function]

FLOPPY.FORMAT erases and initializes the track information on a floppy disk. This must be done when new floppy disks are to be used for the first time. This can also be used to erase the information on used floppy disks.

NAME should be a string that is used as the name of the floppy (106 characters max). This name can be read and set using **FLOPPY.NAME** (below).

If *AUTOCONFIRMFLG* is **NIL**, the user will be prompted to confirm erasing the floppy, if it appears to contain valid information. If *AUTOCONFIRMFLG* is **T**, the user is not prompted to confirm.

If *SLOWFLG* is **NIL**, only the Pilot records needed to give your floppy an empty directory are written. If *SLOWFLG* is **T**, **FLOPPY.FORMAT** will completely erase the floppy, writing track information and critical Pilot records on it. *SLOWFLG* should be set to **T** when formatting a brand-new floppy.

Note: Formatting a floppy is a very compute-intensive operation for the I/O hardware. Therefore, the cursor may stop tracking the mouse and keystrokes may be lost while formatting a floppy. This behavior goes away when the formatting is finished.

Warning: The floppy mode set by **FLOPPY.MODE** (above) affects how **FLOPPY.FORMAT** formats the floppy. If the floppy is going to be used in Pilot mode, it should be formatted under **(FLOPPY.MODE 'PILOT)**. If it is to be used as a CMP floppy, it should be formatted under **(FLOPPY.MODE 'CPM)**. The two types of formatting are incompatible.

(FLOPPY.NAME *NAME*) [Function]

If *NAME* is **NIL**, returns the name stored on the floppy disk. If *NAME* is non-**NIL**, then the name of the floppy disk is set to *NAME*.

(FLOPPY.FREE.PAGES) [Function]

Returns the number of unallocated free pages on the floppy disk in the floppy disk drive.

Note: Pilot floppy files are represented by contiguous pages on a floppy disk. If the user is creating and deleting a lot of files on a floppy, it is advisable to keep such a floppy less than 75 percent full.

(FLOPPY.CAN.READP) [Function]

Returns non-**NIL** if there is a floppy in the floppy drive.

Note: **FLOPPY.CAN.READP** does not provide any debouncing (protection against not fully closing the floppy drive door). It may be more useful to use **FLOPPY.WAIT.FOR.FLOPPY** (below).

(FLOPPY.CAN.WRITEP) [Function]

Returns non-**NIL** if there is a floppy in the floppy drive and the floppy drive can write on this floppy.

It is not possible to write on a floppy disk if the "write-protect notch" on the floppy disk is punched out.

(FLOPPY.WAIT.FOR.FLOPPY *NEWFLG*) [Function]

If *NEWFLG* is **NIL**, waits until a floppy is in the floppy drive before returning.

If *NEWFLG* is **T**, waits until the existing floppy in the floppy drive, if any, is removed, then waits for a floppy to be inserted into the drive before returning.

(FLOPPY.SCAVENGE) [Function]

Attempts to repair a floppy whose critical records have become confused (causing errors when file operations are attempted). May also retrieve accidently-deleted files, provided they haven't been overwritten by new files.

(FLOPPY.TO.FILE *TOFILE*) [Function]

Copies the entire contents of the floppy to the "floppy image" file *TOFILE*, which can be on a file server, local disk, etc. This can be used to create a centralized copy of a floppy, that different users can copy to their own floppy disks (using **FLOPPY.FROM.FILE**).

Note: A floppy image file for an 8-inch floppy is about 2500 pages long, regardless of the number of pages in use on the floppy.

(FLOPPY.FROM.FILE *FROMFILE***)** [Function]

Copies the "floppy image" file *FROMFILE* to the floppy. *FROMFILE* must be a file produced by **FLOPPY.TO.FILE**.

(FLOPPY.ARCHIVE *FILES NAME***)** [Function]

FLOPPY.ARCHIVE formats a floppy inserted into the floppy drive, giving the floppy the name *NAME*#1. **FLOPPY.ARCHIVE** then copies each file in *FILES* to the freshly formatted floppy. If the first floppy fills up, **FLOPPY.ARCHIVE** uses multiple floppies (named *NAME*#2, *NAME*#3, etc.), each time prompting the user to insert a new floppy.

The function **DIRECTORY** (page 24.33) is convenient for generating a list of files to archive. For example,

```
(FLOPPY.ARCHIVE
  (DIRECTORY '{ERIS}<Lisp>Project>*)
  'Project)
```

will archive all files on the directory **{ERIS}<Lisp>Project>** to floppies (named **Project#1**, **Project#2**, etc.).

(FLOPPY.UNARCHIVE *HOST/DIRECTORY***)** [Function]

FLOPPY.UNARCHIVE copies all files on the current floppy to the directory *HOST/DIRECTORY*. For example, **(FLOPPY.UNARCHIVE '{ERIS}<Lisp>Project>)** will copy each file on the current floppy to the directory **{ERIS}<Lisp>Project>**. If there is more than one floppy to restore from archive, **FLOPPY.UNARCHIVE** should be called on each floppy disk.

24.11 I/O Operations to and from Strings

It is possible to treat a string as if it were the contents of a file by using the following function:

(OPENSTRINGSTREAM *STR ACCESS***)** [Function]

Returns a stream that can be used to access the characters of the string *STR*. *ACCESS* may be either **INPUT**, **OUTPUT**, or **BOTH**; **NIL** defaults to **INPUT**. The stream returned may be used exactly like a file opened with the same access, except that output operations may not extend past the end of the original string. Also, string streams do not appear in the value of **(OPENP)**.

For example, after performing

```
(SETQ STRM (OPENSTRINGSTREAM "THIS 2 (IS A LIST)"))
```

the following succession of reads could occur:

```
(READ STRM) => THIS
 (RATOM STRM) => 2
 (READ STRM) => (IS A LIST)
 (EOFP STRM) => T
```

Compatibility Note: In Interlisp-10 it was possible to take input from a string simply by passing the string as the *FILE* argument to an input function. In order to maintain compatibility with this feature, Interlisp-D provides the same capability. This not terribly clean feature persists in the present implementation to give users time to convert old code. This means that strings are *not* equivalent to litatoms when specifying a file name as a stream argument (see page 24.13). In a future release, the old Interlisp-10 string-reading feature will be decommissioned, and **OPENSTRINGSTREAM** will be the only way to perform I/O on a string.

24.12 Temporary Files and the CORE Device

Many operating systems have a notion of "scratch file", a file typically used as temporary storage for data most naturally maintained in the form of a file, rather than some other data structure. A scratch file can be used as a normal file in most respects, but is automatically deleted from the file system after its useful life is up, e.g., when the job terminates, or the user logs out. In normal operation, the user need never explicitly delete such files, since they are guaranteed to disappear soon.

A similar functionality is provided in Interlisp-D by core-resident files. Core-resident files are on the device **CORE**. The directory structure for this device and all files on it are represented completely within the user's virtual memory. These files are treated as ordinary files by all file operations; their only distinguishing feature is that all trace of them disappears when the virtual memory is abandoned.

Core files are opened and closed by name the same as any other file, e.g., **(OPENSTREAM '{CORE}<FOO>FIE.DCOM 'OUTPUT)**. Directory names are completely optional, so files can also have names of the form **{CORE}NAME.EXT**. Core files can be enumerated by **DIRECTORY** (page 24.33). While open, they are registered in **(OPENP)**. They do consume virtual memory space, which is only reclaimed when the file is deleted. Some caution

should thus be used when creating large **CORE** files. Since the virtual memory of an Interlisp-D workstation usually persists far longer than the typical process on a mainframe computer, it is still important to delete **CORE** files after they are no longer in use.

For many applications, the name of the scratch file is irrelevant, and there is no need for anyone to have access to the file independent of the program that created it. For such applications, **NODIRCORE** files are preferable. Files created on the device lisp **NODIRCORE** are core-resident files that have no name and are registered in no directory. These files "disappear", and the resources they consume are reclaimed, when all pointers to the file are dropped. Hence, such files need never be explicitly deleted or, for that matter, closed. The "name" of such a file is simply the stream object returned from **(OPENSTREAM '{NODIRCORE} 'OUTPUT)**, and it is this stream object that must be passed to all input/output operations, including **CLOSEF** and any calls to **OPENSTREAM** to reopen the file.

(COREDEVICE *NAME NODIRFLG***)** [Function]

Creates a new device for core-resident files and assigns *NAME* as its device name. Thus, after performing **(COREDEVICE 'FOO)**, one can execute **(OPENSTREAM '{FOO}BAR 'OUTPUT)** to open a file on that device. Interlisp-D is initialized with the single core-resident device named **CORE**, but **COREDEVICE** may be used to create any number of logically distinct core devices.

If *NODIRFLG* is non-**NIL**, a core device that acts like **{NODIRCORE}** is created.

Compatibility note: In Interlisp-10, it was possible to create scratch files by using file names with suffixes **;S** or **;T**. In Interlisp-D, these suffixes in file names are simply ignored when output is directed to a particular host or device. However, the function **PACKFILENAME.STRING** is defined to default the device name to **CORE** if the file has the **TEMPORARY** attribute and no explicit host is provided.

24.13 NULL Device

The **NULL** device provides a source of content-free "files". **(OPENSTREAM '{NULL} 'OUTPUT)** creates a stream that discards all output directed at it. **(OPENSTREAM '{NULL} 'INPUT)** creates a stream that is perpetually at end-of-file (i.e., has no input).

24.15 Deleting, Copying, and Renaming Files

(DELFILE *FILE***)** [Function]

Deletes *FILE* if possible. The file must be closed. Returns the full name of the file if deleted, else **NIL**. Recognition mode for *FILE* is **OLDEST**, i.e., if *FILE* does not have a version number specified, then **DELFILE** deletes the oldest version of the file.

(COPYFILE *FROMFILE TOFILE***)** [Function]

Copies *FROMFILE* to a new file named *TOFILE*. The source and destination may be on any combination of hosts/devices. **COPYFILE** attempts to preserve the **TYPE** and **CREATIONDATE** where possible. If the original file's file type is unknown, **COPYFILE** attempts to infer the type (file type is **BINARY** if any of its 8-bit bytes have their high bit on).

COPYFILE uses **COPYCHARS** (page 25.20) if the source and destination hosts have different **EOL** conventions. Thus, it is possible for the source and destination files to be of different lengths.

(RENAMEFILE *OLDFILE NEWFILE***)** [Function]

Renames *OLDFILE* to be *NEWFILE*. Causes an error, **FILE NOT FOUND** if *FILE* does not exist. Returns the full name of the new file, if successful, else **NIL** if the rename cannot be performed.

If *OLDFILE* and *NEWFILE* are on the same host/device, and the device implements a renaming primitive, **RENAMEFILE** can be very fast. However, if the device does not know how to rename files in place, or if *OLDFILE* and *NEWFILE* are on different devices, **RENAMEFILE** works by copying *OLDFILE* to *NEWFILE* and then deleting *OLDFILE*.

24.16 Searching File Directories

DIRECTORIES [Variable]

Global variable containing the list of directories searched (in order) by **SPELLFILE** and **FINDFILE** (below) when not given an explicit *DIRLST* argument. In this list, the atom **NIL** stands for the login directory (the value of **LOGINHOST/DIR**), and the atom **T** stands for the currently connected directory. Other elements should be *full* directory specifications, e.g., **{TWENTY}PS:<LISPUSERS>**, not merely **LISPUSERS**.

LISPUSERSDIRECTORIES [Variable]

Global variable containing a list of directories to search for "library" package files. Used by the **FILES** file package command (page 17.39).

(SPELLFILE *FILE NOPRINTFLG NSFLG DIRLST***)** [Function]

Searches for the file name *FILE*, possibly performing spelling correction (see page 20.15). Returns the corrected file name, if any, otherwise **NIL**.

If *FILE* has a directory field, **SPELLFILE** attempts spelling correction against the files in that particular directory. Otherwise, **SPELLFILE** searches for the file on the directory list *DIRLST* before attempting any spelling correction.

If *NOPRINTFLG* is **NIL**, **SPELLFILE** asks the user to confirm any spelling correction done, and prints out any files found, even if spelling correction is not done. If *NOPRINTFLG* = **T**, **SPELLFILE** does not do any printing, nor ask for approval.

If *NSFLG* = **T** (or **NOSPELLFLG** = **T**, see page 20.13), no spelling correction is attempted, though searching through *DIRLST* still occurs.

DIRLST is the list of directories searched if *FILE* does not have a directory field. If *DIRLST* is **NIL**, the value of the variable **DIRECTORIES** is used.

Note: If *DIRLST* is **NIL**, and *FILE* is not found by searching the directories on **DIRECTORIES**, but the root name of *FILE* has a **FILEDATES** property (page 17.20) indicating that a file by that name has been loaded, then the directory indicated in the **FILEDATES** property is searched, too. This additional search is not done if *DIRLST* is non-**NIL**.

ERRORTYPELST (page 14.22) initially contains the entry **((23 (SPELLFILE (CADR ERRORMESS) NIL NOFILESPELLFLG)))**, which causes **SPELLFILE** to be called in case of a **FILE NOT FOUND** error. If the variable **NOFILESPELLFLG** is **T** (its initial value), then spelling correction is not done on the file name, but **DIRECTORIES** is still searched. If **SPELLFILE** is successful, the operation will be reexecuted with the new (corrected) file name.

(FINDFILE *FILE NSFLG DIRLST***)** [Function]

Uses **SPELLFILE** to search for a file named *FILE*. If it finds one, returns its full name, with no user interaction. Specifically, it calls **(SPELLFILE** *FILE* **T** *NSFLG DIRLST***)**, after first performing two simple checks: If *FILE* has an explicit directory, it checks to see if a file so named exists, and if so returns that file. If *DIRLST* is **NIL**, it looks for *FILE* on the connected directory before calling **SPELLFILE**.

24.17 Listing File Directories

The function **DIRECTORY** allows the user to conveniently specify and/or program a variety of directory operations:

(DIRECTORY *FILES COMMANDS DEFAULTEXT DEFAULTVERS***)** [Function]

Returns, lists, or performs arbitrary operations on all files specified by the "file group" *FILES*. A file group has the form of a regular file name, except that the character * can be used to match any number of characters, including zero, in the file name. For example, the file group **A*B** matches all file names beginning with the character **A** and ending with the character **B**. The file group ***.DCOM** matches all files with an extension of **DCOM**.

If *FILES* does not contain an explicit extension, it is defaulted to *DEFAULTEXT*; if *FILES* does not contain an explicit version, it is defaulted to *DEFAULTVERS*. *DEFAULTEXT* and *DEFAULTVERS* themselves default to *. If the period or semicolon preceding the omitted extension or version, respectively, is present, the field is explicitly empty and no default is used. All other unspecified fields default to *. Null version is interpreted as "highest". Thus *FILES* = * or *.* or *.*;* enumerates all files on the connected directory; *FILES* = *. or *.;* enumerates all versions of files with null extension; *FILES* = *.; enumerates the highest version of files with null extension; and *FILES* = *.*; enumerates the highest version of all files. If *FILES* is **NIL**, it defaults to *.*;*.

Note: Some hosts/devices are not capable of supporting "highest version" in enumeration. Such hosts instead enumerate *all* versions.

For each file that matches the file group *FILES*, the "file commands" in *COMMANDS* are executed in order. Some of the file commands allow aborting the command processing for a given file, effectively filtering the list of files. The interpretation of the different file commands is described below. If *COMMANDS* is **NIL**, it defaults to **(COLLECT)**, which collects the matching file names in a list and returns it as the value of **DIRECTORY**.

The "file commands" in *COMMANDS* are interpreted as follows:

P Prints the file's name. For readability, **DIRECTORY** strips the directory from the name, printing it once as a header in front of each set of consecutive files on the same directory.

PP Prints the file's name without a version number.

a string Prints the string.

READDATE, WRITEDATE
CREATIONDATE, SIZE
LENGTH, BYTESIZE
PROTECTION, AUTHOR
TYPE Prints the appropriate information returned by **GETFILEINFO** (page 24.17).

COLLECT Adds the full name of this file to an accumulating list, which will be returned as the value of **DIRECTORY**.

COUNTSIZE Adds the size of this file to an accumulating sum, which will be returned as the value of **DIRECTORY**.

DELETE Deletes the file.

DELVER If this file is not the highest version of files by its name, delete it.

PAUSE Waits until the user types any char before proceeding with the rest of the commands (good for display if you want to ponder).

The following commands are predicates to filter the list. If the predicate is not satisfied, then processing for this file is aborted and no further commands (such as those above) are executed for this file.

Note: if the **P** and **PP** commands appear in *COMMANDS* ahead of any of the filtering commands below except **PROMPT**, they are postponed until after the filters. Thus, assuming the caller has placed the attribute options after the filters as well, no printing occurs for a file that is filtered out. This is principally so that functions like **DIR** (below) can both request printing and pass arbitrary commands through to **DIRECTORY**, and have the printing happen in the appropriate place.

PROMPT *MESS* Prompts with the yes/no question *MESS*; if user responds with **No**, abort command processing for this file.

OLDERTHAN *N* Continue command processing if the file hasn't been referenced (read or written) in *N* days. *N* can also be a string naming an explicit date and time since which the file must not have been referenced.

NEWERTHAN *N* Continue command processing if the file has been written within the last *N* days. *N* can also be a string naming an explicit date and time. Note that this is not quite the complement of **OLDERTHAN**, since it ignores the read date.

BY *USER* Continue command processing if the file was last written by the given user, i.e., its **AUTHOR** attribute matches (case insensitively) *USER*.

@ *X* *X* is either a function of one argument (*FILENAME*), or an arbitrary expression which uses the variable **FILENAME** freely. If *X* returns **NIL**, abort command processing for this file.

The following two commands apply not to any particular file, but globally to the manner in which directory information is printed.

OUT *FILE* Directs output to *FILE*.

COLUMNS *N* Attempts to format output in *N* columns (rather than just 1).

DIRECTORY uses the variable **DIRCOMMANDS** as a spelling list to correct spelling and define abbreviations and synonyms (see page 20.15). Currently the following abbreviations are recognized:

AU	**=>**	**AUTHOR**
-	**=>**	**PAUSE**
COLLECT?	**=>**	**PROMPT " ? " COLLECT**
DA **DATE**	**=>**	**CREATIONDATE**
TI	**=>**	**WRITEDATE**
DEL	**=>**	**DELETE**
DEL? **DELETE?**	**=>**	**PROMPT " delete? " DELETE**
OLD	**=>**	**OLDERTHAN 90**
PR	**=>**	**PROTECTION**
SI	**=>**	**SIZE**
VERBOSE	**=>**	**AUTHOR CREATIONDATE SIZE READDATE WRITEDATE**

(FILDIR *FILEGROUP***)** [Function]

Obsolete synonym of **(DIRECTORY** *FILEGROUP***)**.

(DIR *FILEGROUP* COM_1 ... COM_N**)** [NLambda NoSpread Function]

Convenient form of **DIRECTORY** for use in type-in at the executive. Performs **(DIRECTORY '***FILEGROUP* **'(P** COM_1 ... COM_N**))**.

(NDIR *FILEGROUP* COM_1 ... COM_N**)** [NLambda NoSpread Function]

Version of **DIR** that lists the file names in a multi-column format. Also, by default only lists the most recent version of files (unless *FILEGROUP* contains an explicit version).

24.18 File Servers

A file server is a shared resource on a local communications network which provides large amounts of file storage. Different file servers honor a variety of access protocols. Interlisp-D supports the following protocols: PUP-FTP, PUP-Leaf, and NS Filing. In addition, there are library packages available that support other communications protocols, such as TCP/IP and RS232.

With the exception of the RS232-based protocols, which exist only for file transfer, these network protocols are integrated into the Interlisp-D file system to allow files on a file server to be treated in much the same way files are accessed on local devices, such as the disk. Thus, it is possible to call **OPENSTREAM** on the file **{ERIS}<LISP>FOO.DCOM;3** and read from it or write to it just as if the file had been on the local disk (**{DSK}<LISP>FOO.DCOM;3**), rather than on a remote server named **ERIS**. However, the protocols vary in how much control they give the workstation over file system operations. Hence, some restrictions apply, as described in the following sections.

24.18.1 Pup File Server Protocols

There are two file server protocols in the family of Pup protocols: Leaf and FTP. Some servers support both, while others support only one of them. Interlisp-D uses whichever protocol is more appropriate for the requested operation.

Leaf is a random access protocol, so files opened using these protocols are **RANDACCESSP** (page 25.20), and thus most normal i/o operations can be performed. However, Leaf does not support directory enumeration. Hence, **DIRECTORY** cannot be used on a Leaf file server unless the server also supports FTP. In addition, Leaf does not supply easy access to a file's attributes. **INFILEP** and **GETFILEINFO** have to open the file for input in order to obtain their information, and hence the file's read date will change, even though the semantics of these functions do not imply it.

FTP is a file transfer protocol that only permits sequential access to files. However, most implementations of it are considerably more efficient than Leaf. Interlisp-D uses FTP in preference to Leaf whenever the call to **OPENSTREAM** requests sequential access only. In particular, the functions **SYSOUT** and **COPYFILE** open their files for sequential access. If a file server supports FTP but for some reason it is undesirable for Lisp to use it, one can set the internal variable **\FTPAVAILABLE** to **NIL**.

The system normally maintains a Leaf connection to a host in the background. This connection can be broken by calling

(**BREAKCONNECTION** *HOST*). Any subsequent reference to files on that host will reestablish the connection. The principal use for this function arises when the user interrupts a file operation in such a way that the file server thinks the file is open but Lisp thinks it is closed (or not yet open). As a result, the next time Lisp tries to open the file, it gets a file busy error.

24.18.2 Xerox NS File Server Protocols

Interlisp supports file access to Xerox 803x file servers, using the Filing Protocol built on Xerox Network Systems protocols. Interlisp-D determines that a host is an NS File Server by the presence of a colon in its name, e.g., **{PHYLEX:}**. The general format of NS fileserver device names is {*SERVERNAME*:*DOMAIN*:*ORGANIZATION*}; the device specification for an 8000-series product in general includes the ClearingHouse domain and organization. If domain and organization are not supplied directly, then they are obtained from the defaults, which themselves are found by consulting the nearest ClearingHouse if the user has not defined them in an init file (page 31.8). However, note that the server name must still have a colon in it to distinguish it from other types of host names (e.g., Pup server names).

NS file servers in general permit arbitrary characters in file names. The user should be cognizant of file name quoting conventions (page 24.6), and the fact that any file name presented as a litatom needs to have characters of significance to the reader, such as space, escaped with a %. Of course, one can always present the file name as a string, in which case only the quoting conventions are important.

NS file servers support a true hierarchical file system, where subdirectories are just another kind of file, which needs to be explicitly created. In Interlisp, subdirectories are created automatically as needed: A call to **OPENFILE** to create a file in a non-existent subdirectory automatically creates the subdirectory; **CONN** to a non-existent subdirectory asks the user whether to create the directory. For those using Star software, a directory corresponds to a "File Drawer", while a subdirectory corresponds to a "File Folder".

Because of their hierarchical structure, NS directories can be enumerated to arbitrary levels. The default is to enumerate all the files (the leaves of the tree), omitting the subdirectory nodes themselves. This default can be changed by the following variable:

FILING.ENUMERATION.DEPTH [Variable]

This variable is either a number, specifying the number of levels deep to enumerate, or **T**, meaning enumerate to all levels. In the former case, when the enumeration reaches the specified depth, only the subdirectory name rooted at that level is listed, and none of its descendants is listed. When **FILING.ENUMERATION.DEPTH** is **T**, all files are listed, and no subdirectory names are listed. **FILING.ENUMERATION.DEPTH** is initially **T**.

Independent of **FILING.ENUMERATION.DEPTH**, a request to enumerate the top-level of a file server's hierarchy lists only the top level, i.e., assumes a depth of 1. For example, **(DIRECTORY '{PHYLEX:})** lists exactly the top-level directories of the server **PHYLEX:**.

NS file servers do not currently support random access. Therefore, **SETFILEPTR** of an NS file generally causes an error. However, **GETFILEPTR** returns the correct character position for open files on NS file servers. In addition, **SETFILEPTR** works in the special case where the file is open for input, and the file pointer is being set forward. In this case, the intervening characters are automatically read.

Even while Interlisp has no file open on an NS Server, the system maintains a "session" with the server for a while in order to improve the speed of subsequent requests to the server. While this session is open, it is possible for some nodes of the server's file system to appear "busy" or inaccessible to certain clients on other workstations (such as Star). If this happens, the following function can be used to terminate any open sessions immediately:

(BREAK.NSFILING.CONNECTION *HOST***)** [Function]

Closes any open connections to NS file server *HOST*.

24.18.3 Operating System Designations

Some of the network server protocols are implemented on more than one kind of foreign host. Such hosts vary in their conventions for logging in, naming files, representing end-of-line, etc. In order for Interlisp to communicate gracefully with all these hosts, it is necessary that the variable **NETWORKOSTYPES** be correctly set.

NETWORKOSTYPES [Variable]

An association-list that associates a host name with its operating system type. Elements in this list are of the form (*HOSTNAME* .

TYPE), for example, **(MAXC2 . TENEX)**. The operating system types currently known to Lisp are **TENEX**, **TOPS20**, **UNIX**, and **VMS**. The host names in this list should be the "canonical" host name, represented as an uppercase atom. For Pup and NS hosts, the function **CANONICAL.HOSTNAME** (below) can be used to determine which of several aliases of a server is the canonical name.

(CANONICAL.HOSTNAME *HOSTNAME*) [Function]

Returns the "canonical" name of the server *HOSTNAME*, or **NIL** if *HOSTNAME* is not the name of a server.

24.18.4 Logging In

Most file servers require a user name and password for access. Interlisp-D maintains an ephemeral database of user names and passwords for each host accessed recently. The database vanishes when **LOGOUT**, **SAVEVM**, **SYSOUT**, or **MAKESYS** is executed, so that the passwords remain secure from any subsequent user of the same virtual memory image. Interlisp-D also maintains a notion of the "default" user name and password, which are generally those with which the user initially logs in (on the 1132, the default user name corresponds to that displayed in the Alto executive).

When a file server for which the system does not yet have an entry in its password database requests a name and password, the system first tries the default user name and password. If the file server doesn't recognize that name/password, the system prompts the user for a name and password to use for that host. It suggests a default name:

{ERIS} Login: Green

which the user can accept by typing a carriage return, or replace the name by typing a new name or backspacing over it. Following the name, the user is prompted for a password:

{ERIS} Login: Verdi (password)

which is not echoed, terminated by another carriage return. This information is stored in the password database so that the user is prompted only once, until the database is again cleared.

Interlisp-D also prompts for password information when a protection violation occurs on accessing a directory on certain kinds of servers that support password-protected directories. Some such servers allow one to protect a file in a way that it is inaccessible to even its owner until the file's protection is changed; in such case, no password would help, and the system causes the normal **PROTECTION VIOLATION** error.

The user can abort a password interaction by typing the **ERROR** interrupt, initially Control-E. This generally either causes a **PROTECTION VIOLATION** error, if the password was requested in order to gain access to a protected file on an otherwise accessible server; or to act as though the server did not exist, in the case where the password was needed in order to gain *any* access to the server.

The following functions are useful for altering the password database:

(LOGIN *HOSTNAME FLG DIRECTORY MSG***)** [Function]

Forces Interlisp-D to ask for the user name and password to be used when accessing host *HOSTNAME*. Any previous login information for *HOSTNAME* is overriden. If *HOSTNAME* is **NIL**, it overrides login information for all hosts and resets the default user name and password to be those typed in by the user. The special value *HOSTNAME* = **NS::** is used to obtain the default user name and password for all logins for NS Servers.

If *FLG* is the atom **QUIET**, only prompts the user if there is no cached information for *HOSTNAME*.

If *DIRECTORY* is specified, it is the name of a directory on *HOSTNAME*. In this case, the information requested is the "connect" password for that directory. Connect passwords for any number of different directories on a host can be maintained.

If *MSG* is non-**NIL**, it is a message (a string) to be printed before the name and password information is requested.

LOGIN returns the user name with which the user completed the login.

(SETPASSWORD *HOST USER PASSWORD DIRECTORY***)** [Function]

Sets the values in the internal password database, exactly as if the strings *USER* and *PASSWORD* were typed in via **(LOGIN** *HOST* **NIL** *DIRECTORY***)**.

(SETUSERNAME *NAME***)** [Function]

Sets the default user name to *NAME*.

(USERNAME *FLG STRPTR PRESERVECASE***)** [Function]

If *FLG* = **NIL**, returns the default user name. This is the only value of *FLG* that is meaningful in Interlisp-D.

USERNAME returns the value as a string, unless *STRPTR* is **T**, in which case **USERNAME** returns the value as an atom. The name is returned in upper case, unless *PRESERVECASE* is true.

24.18.5 Abnormal Conditions

If Interlisp-D tries to access a file and does not get a response from the file server in a reasonable period of time, it prints a message that the file server is not responding, and keeps trying. If the file server has actually crashed, this may continue indefinitely. A control-E or similar interrupt aborts out of this state.

If the file server crashes but is restarted before the user attempts to do anything, file operations will usually proceed normally, except for a brief pause while Interlisp-D tries to reestablish any connections it had open before the crash. However, this is not always possible. For example, when a file is open for sequential output and the server crashes, there is no way to recover the output already written, since it vanished with the crash. In such cases, the system will cause an error such as **Connection Lost**.

LOGOUT closes any file server connections that are currently open. On return, it attempts to reestablish connections for any files that were open before logging out. If a file has disappeared or been modified, Interlisp-D reports this fact. Files that were open for sequential access generally cannot be reopened after **LOGOUT**.

Interlisp supports simultaneous access to the same server from different processes and permits overlapping of Lisp computation with file server operations, allowing for improved performance. However, as a corollary of this, a file is not closed the instant that **CLOSEF** returns; Interlisp closes the file "in the background". It is therefore very important that the user exits Interlisp via **(LOGOUT)**, or **(LOGOUT T)**, rather than boot the machine.

On rare occasions, the Ethernet may appear completely unresponsive, due to Interlisp having gotten into a bad state. Typing **(RESTART.ETHER)** will reinitialize Lisp's Ethernet driver(s), just as when the Lisp system is started up following a **LOGOUT**, **SYSOUT**, etc (see page 31.38)