F.G.H.
3/21/85

LispCourse #15: Files and Directories ž Part 1

Basic Concepts

What is a file?

A file is a "place" where you keep information.

A file is the basic unit of permanent external storage in Interlisp.

A file is a bunch of information or data that should be manipulated in one chunk.

A file is the basic organizational construct for storing data in Interlisp.

???

File Names

Every file has a name. In general, file names have two parts: a *name* and an *extension.* Both the name and the extension are litatoms with no spaces or other unusual characters such as Tabs, ":", ".", ","etc.

In the file name, the name and the extension are seprated by a period ("."). For example, *FILE.EXT* is a file name. So is *NewData3-3-85.TED.* In this latter example, *NewData3-3-85* is the name part and *TED* is the extension part.

The extension part is optional. *XYZZY* is a perfectly good file name.

The name part of the file name is usually used to indicate what the file contains. The extension part is usually indicative of the flavor of the file. (See discussion of flavors below.)

Note that Interlisp is NOT case sensitive with respect to file names. The file FOO.BAR and the file Foo.Bar are the SAME file according to Interlisp.

Flavors of files

From one point of view, all files are alike: they are an ordered stream of bits (1s and 0s) that must be stored somewhere for later retrieval.

From another point of view, there are many different *flavors* of files.

The pattern of 1s and 0s in a file is really a code that stands for some meaningful data or information.

In order to extract the meaningful information in a file, we need to know how to interpret the format of the bits in that file.

Flavors of files arise because different files contain different kinds of meaningful information and represent that information using differrent codes and patterns of bits.

Examples:

TEdit files contain formatted text.

Bravo files also contain formatted text
(but using a different "code" than TEdit files)

Lafite mail files contain a sequence of mail messages

DCOM files contain compiled Lisp programs

SYSOUT files contain Interlisp virtual memories

Each of these types of file contains a different kind of information and/or represents information using a different code.

You alternate between these two viewpoints when dealing with files in Interlisp, depending on whether you are dealing with the *file* itself or with the *information in the file.*

A function like COPYFILE deals with the *file itself* and therefore treats a file like a stream of bits to be copied from one location to another. Thus COPYFILE is very general and can operate on files of any flavor.

Functions like TEdit, LOAD, and LAFITE make use of the *information within a file* and therefore work only with files of certain flavors. They will totally misinterpret files of the wrong flavor. Or worse, they often crash when processing files of the wrong flavor.

The flavor of a file is (in Interlisp) *intensionally* defined. There is no way (other than naming conventions) to specifically declare a file to be of a particular flavor.

The system itself treats all files as bit streams. This bit stream will be interpreted as meaningful information or as garbage depending on what function you use to access it.

For example, you can access a DCOM file using TEdit, but it will appears as (mostly) garbage in the TEdit window. Similarly, you can read in a TEdit file using LOAD, but in all probablilty the LOAD will bomb in a very short time.

Its up to each user to use TEdit on files that are meant to be TEdit files and to use LOAD on files that are meant to be Lisp files.

The system provides only minimal help in helping you with this task!! The most you can expect is that a function will detect and informa you that you have asked it to work on a fuile it can't interpret.

To overcome the lack of the distinction between flavors of files, the *convention* has been established that the extension of each file name (see discussion of file names below) gives some indication of the file's flavor.

Examples:

TEdit files end in .TED or .TEDIT

DCOM files end in .DCOM

SYSOUT files end in .SYSOUT

Lafite files end in .MAIL

Devices

The stream of bits that makes up a file has to be stored on some *"physical"* device such as a floppy, a disk, a file server, etc.

Interlisp supports file storage on many different kinds of devices.

Interlisp also tries to make file storage on all of the different devices behave as similarly as possible.

But different devices have different characteristics, both hardware and software, and Interlisp cannot always cover up these differences.

So. How you use files in Interlisp depends a little bit on what device the file is stored on.

Moreover, as you move a file from device to device, its characteristics may change slightly.

I will try to point out these device-dependencies as we go along.

The devices supported by Interlisp-D are the following:

Local Disk

DLion local disk ž 1 or more logical volumes used for Lisp files

Dolphin local disk ž 2 Alto partitions

Dorado local disk ž 5 or 19 Alto partitions

File Servers

IFS ž e.g., Phylum, Eris, etc.

NS ž e.g., LispFiles:, StarFiles:

Vax (Unix) ž e.g., PARC-CSLI, PARC-VAXC

Floppy Disks (Dlion only)

Core Devices (simulated devices in the Lisp virtual memory)

Device Names:

All devices have a name. When used to refer to a file, the name of the device is enclosed in curly-brackets. The device name conventions are as follows:

Local Disk

DLion local disk ž {DSK}

Dolphin local disk ž {DSK}, {DSK1}, {DSK2}

Dorado local disk ž {DSK}, {DSK*i*}

File Servers

IFS ž the name of the server, e.g., {Phylum}, {Eris}, etc.

NS ž the NS network adderess of the server or its abbreviation as in {LispFiles:PARC:Xerox} or {LispFiles:} for short. Note the abbreviated name always ends in a ":".

Vax (Unix) ž the name of the machine, e.g., {PARC-CSLI}, {PARC-VAXC}

Floppy Disks ž {FLOPPY}

Core Devices ž more than one core device can be created using the function COREDEVICE. The name of the core device is set by this function. See p. 18.13 of the IRM. The device {CORE} is already created in every system.

Directories

Different devices organize the files they contain in different ways.

Some devices use a **flat** organization, i.e., the files are just stored on the device one after the other. On some devices the files are ordered (e.g., alphabetically). On some devices the files are randomly ordered.

Example:

```
  {DSK}
ACCOUNTANT.RUN;1
ALTOD0MC.EB;1
ALTOD1MC.EB;1
CHAT.RUN;1
Com.cm;1
DiskDescriptor.;1
DoradoLisp.MB;1
DORADOLISPMC.EB;1
Dumper.boot;1
EMPRESS.RUN;1
Executive.Run;1
Fonts.Widths;1
FOOTNOTES.TEDIT;2
FOOTNOTES.TEDIT;1
Ftp.log;1
Ftp.Run;1
INIT.LISP;1
LISP.RUN;1
LISP.SYMS;1
LISP.VIRTUALMEM;1
Rem.Cm;1
Swat.;1
Swatee.;1
Sys.Boot;1
SYS.ERRORS;1
SysDir.;1
SysFont.Al;1
```

User.Cm;1

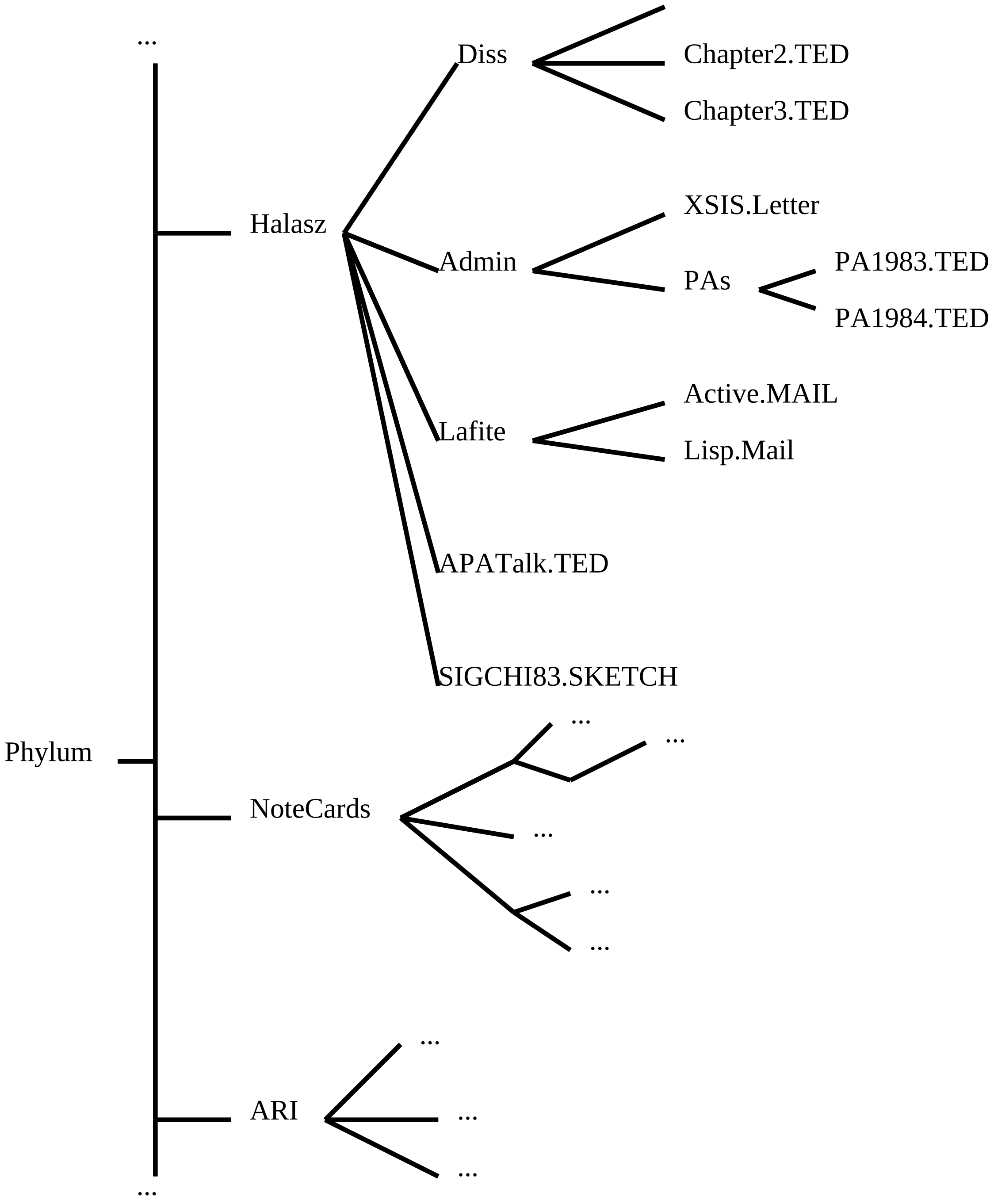
However, most devices use **directories** and **subdirectories** to organize the files they contain into logical groupings:

> Basically a **directory** is just a *list* of 1 or more files that are stored together in the same logical "place" on the device.
>
> A **subdirectory** is a *list* of 1 or more files from a single directory that are stored together in the same logical "place" within that *directory*.

Every directory and sub-directory has a name, which is usually a short atom.

Directories and subdirectories organize the files on a device into a tree structure. The device is divided into a set of directories. Each directory is sub-divided into zero or more sub-directories. Each sub-directory is in turn divided into zero or more sub-sub-directories. Files can be placed in any directory or sub-directory in this tree structure.

Diagramatically:



Path Names

The directory or sub-directory that a file is located in can be uniquely specified by a path that starts at the top-level directory and includes each sub-directory in turn until the file is reached in the tree structure.

When refering to this path name, the directory name is enclosed in angle brackets, while each sub-directory is terminated by an end angle bracket.

Examples:

PA1984.TED is in <Halasz>Admin>PAs>

APA.Talk is in <Halasz>

Lisp.mail is in <Halasz>Lafite>

Creating Directories/Sub-directories:

On some devices directories and sub-directories have to be explicitly created before any files can be entered into them. The method for creating directories and sub-directories differs between devices. On some devices, e.g., IFSs, it requires an administrator with special priveleges to create a new directory.

Directories that are created explicitly remain even if they have no files in them, e.g., after the last file in the directory/sub-directory has been deleted.

On devices that don't require directories and/or sub-directories to be explicitly created, you can create directories and/or sub-directories simply by creating a file that has the new directory/sub-directory as part of its name. The directory and/or sub-directory will be created when the file is created.

Directories/sub-directories that are created implicitly by naming a file disappear after the last file with the directory/sub-directory in its name is deleted.

The various devices and their directory types are as follows:

Local Disk

DLion ž directories are logical volumes on disk created during intial disk partitioning; sub-directories are created implictly by naming.

Dolphin/Dorado ž each Alto parition is a separate device (named DSK1 thru DSK*n*). No directories are supported within these devices, i.e., each device has a FLAT structure.

File Servers

IFS ž directories are created explicitly by an IFS adminstrator, subdirectories are created implicitly by naming.

NS ž directories (called file drawers) are created explicitly by an administrator, sub-directories are created explicitly by naming. Note, however, deleting the last file from a sub-directory DOES NOT delete the sub-directory.

Vax (Unix) ž directories are created explicitly by an administrator, sub-directories are created explicitly by the user. To create a sub-directory, you must log into the Vax and use the Unix mkdir command.

Floppy Disks ž support either a flat file structure or a hierarchical (directory/sub-directory) file structure or both. Both directories and sub-directories are created implicitly by naming.

Core Devices ž support either a flat file structure or a hierarchical (directory/sub-directory) file structure or both. Both directories and sub-directories are created implicitly by naming.

Versions

Interlisp supports file **versions.** That is, Interlisp allows two files with the same name to be on the same device in the same directory/sub-directory.

When this happens, the files are considered to be different *versions* of the same file.

Every file has a **version number**. When a file of a given name in a given device/directory is first created, it is given a version number of 1. When a new version is created, it is assigned a version number one higher than the highest version already in existence.

In Interlisp, the version number is specified after the file name, preceded by a semi-colon. For example, *TESTFILE.TED;3* and *TESTFILE.TED;4* are two versions of the same file. *NewFile;6* and *NewFile;8* are two versions of another file.

Different versions of the "same" file are actually totally separate files and could be treated as such. Version1 of a file could be a TEdit file and version 2 a DCOM file.

However, this would violate the *intent* of versioning. Different versions of a file are supposed to be different versions (e.g., updates) of the "same" information. Lots of little things in Interlisp support this notion.

Examples:

When you name a file without a version number, Interlisp always thinks you mean the latest version of the file. To get to an earlier version of a file, you have to specify the exact version number. (Execption is in deleteing a file, where Interlisp always assumnes you mean the lowest version number unless otherwise specified.)

When you PUT a file, TEdit just writes a new version of the file. The old (unedited) version still exists.

When you do a MAKEFILE, the system just writes a new version of the file. The old (i.e., previous) version still exists and can still be LOADed by specifiying its version number in the LOAD command.

Full File Names

A file is uniquely specified by a full file name containing the name of the device its on, its path name, its file name and its version.

For example, *{phylum}<halasz>lisp>init.lisp;4* is a full file name. *{DSK}FOO.TED;3* and *{FLOPPY}<FOO>BAZ;55* are also full file names.

Ultimately ALL file references in Interlisp MUST specify a full file name. If this wasn't the case, there would be ambiguity as to which file was being referenced.

However, Interlisp allows elliptical references to files that allow you to use much less than the full file name when refering to a file.

> For example, if a version number is not specified, Interlisp assumes you mean the highest version number.
>
> Also, Interlisp will use your connected directory (see the CONN command below) if you don't specify a device and directory.
>
> > Thus, the name *FOO.BAR* may be an allowable abbreviation of *{phylum}<halasz>lisp>foo.bar;5* if you are connected to {phylum}<halasz>lisp> and 5 is the highest version of FOO.BAR in {phylum}<halasz>lisp>.
>
> Note: it almost NEVER hurst to specify the full file name of a file if you know it.

Reminder Note: Interlisp is not case-sensitive with regard to file names.

File Attributes

In addition to their name, all files have some attributes. The attributes attached to a file depend somewhat on the device the file is on. However, in general a file has the following attributes:

SIZE ž the size of the file in disk pages (512 bytes each)

LENGTH ž the length of the file in bytes (~ 512 times its size).

AUTHOR ž login name of person who created this file

CREATIONDATE ž the date and time of the creation of the file.

READDATE ž the date and time when this file was last read.

WRITEDATE ž the date and time when this file was last written to.

TYPE ž text or binary (to help along some older computers and file servers for which this was an important distinction)

BYTESIZE ž in Interlisp-D always 8, but some older computers and file servers allow other sizes.

The attributes of a file can be seen using the FILEBROWSER or the DIR command. Both of these are described below.

You can also access the attributes of a file one at a time using the function GETFILEINFO which has two arguments: the (full) file name and the name of the attribute you want to see.

Example:

```
10_ (GETFILEINFO '<halasz>lisp>init 'SIZE)
15
11_
```

In general, you do not change the attributes of a file directly. The attributes change as you work with the file, e.g., as you add information to the file, its size increases.

File Protection

Some devices allow an additional set of attributes for a file: *the protection code*. The protection code of a file determines who can read or write the file.

Protection schemes vary from device to device.

On the Local Disk, Floppy and Core devices there are no protection codes for files. Basically, anyone can read or write onto any file on these devices.

> There is some protection for Dolphin/Dorado local disks since whole Alto partitions are passworded. You cannot read from or write to any file on an Alto partition (i.e., on {DSK*i*} for any *i*) unless you can supply the password for that partition. However, once you give the correct password, all the files on that partitin can be accessed at will.

On the IFS file servers, a file has three separate protections:

a *read* protection ž who can read from this file

a *write* protection žwho can write on this file

an *append* protection ž who can append something to the end of this file

Each of these is a list of people or distribution lists who have the permission carry out the specified action. The list is either a person's grapevine name (e.g., Halasz.pa), a grapevine distribution list (e.g., USR^.pa), the atom *owner* (indicating the AUTHOR of the file), or the atom *USRegistries^.Internet* (basically indicating anyone with access to the Xerox network).

For example, *{phylum}<halasz>lisp>init* has the protection:

R: USRegistries^.Internet Owner; W: Owner; A: Owner

> This means that any one in the Xerox world can read this file but only I (as owner of the file) can write or append to it.

But, *{ERIS}<LispCore>next>full.sysout* has the protection:

R: Lispcore^.pa LispCoreAccess^.pa Owner; W: Lispcore^.pa Owner; A: Lispcore^.pa Owner

This means that any one on the distribution lists called Lispcore^.pa and LispCoreAccess^.pa can read this file but only people on Lispcore^.pa (and its owner whoever it may be) can write or append to it.

To list or change the protection of a file on an IFS, you have to CHAT to the IFS.

To see the protection codes of a file type:

List *FileName*,<RETURN>

The IFS will respond with a @@ prompt.

Type:

Protection<RETURN><RETURN>

This will list the file with its protection codes as above.

To change the protection codes type:

Change Protection *FileName*<RETURN>

The IFS will respond with the @@ prompt.

Type:

Write *PersonOrDistList*<RETURN><RETURN>

This will add the designated person/group to the WRITE permission list of the file. Use *Read* or *Append* instead of *Write* to change the Read and Append protections. If the Read, Write, or Append is preceded by a *No*, then the designated group/person is remioved from the permission list. (E.g., *No Read LispCore^.pa*).

The Vax and NS file servers have similar, but different protection schemes. See you file server administartor for information on the relevant protection schemes and how to use them.

References

Files are covered in Chapter 6 and Sections 18.16 and 18.17 of the IRM. User relevant material is scattered throughout these sections, so you’ll just have to skim for what you’re interested in.

The file protection scheme for the IFS is covered in the "How to use the IFS" memo on {indigo}<ifs>howtouse.bravo (or .press).