bcc	title	prefix/class-number.revision
	SYSTEM I FILE-NAMING SYSTEM	FNS/W-4

checked	authors	approval date / revision date
Butler W Lampson	Larry L. Barnes Larry L Barnes	
checked		classification
Charles Thacker		Working Paper
approved		distribution / pages
Mef		Company Private / 11

ABSTRACT and CONTENTS

This document specifies the operation of the file-naming component of the utility system.

TABLE OF CONTENTS

I The Structure of File Names

File names, like all other names, will be restricted in length and character set. A maximum length of 16 characters for the main name and 4 characters of type seem adequate. The character set will be restricted to letters, digits, and a small subset of non-alphanumeric ASCII graphics. Lower case letters will be converted to upper case.

The structure of a file name is given by the following productions.

```
file:name = [access:key:name ":" ]name:spec

name:spec = [user:spec] main:name [":" type]
access:key:name = [user:spec] ">" main:name

user:spec = "#" user:number ":" / "(" user:name ":"

main:name = name / "'" name "'"
type = name
user:name = name

user:number = 1$ digit

name = ichars [word] $ ("-" word)
word = achar $ achar

ichars = letter / "$" / "*" / "/" / "?"
achar = ichar / digit
```

The system will adopt the following policies with respect to the use of file types, both in the name string and in the name look-up call. If the type is given in the name:spec, then the file corresponding to the name:spec will be opened or created for output calls. Otherwise, the type in the call will be used to complete the name:spec, if that type is sup-

plied. And if no type is specified in either the string or call, there must be only one file with the designated main: name in the file directory(MIB). The file type for the selected file will be returned when the name is collected.

When a file:name is collected, only the main:names may be abbreviated. User:names, account:names, and types must be given in full. The abbreviation look-up is described in section 4. The main:name may be surrounded with double quotes if a new name is desired. In any case a quoted name suppresses the abbreviation mechanism.

II Structure of File Types

The following proposal for a system type-structure will be implemented. We assume the existence of a set of pre-known file types including: symbolic (S), language symbolic (LS), symbolic data (DS), binary data (DB), binary program (LB),etc. We also assume the existence of an arbitrary set of user-defined and processor-defined file types, for example, FORTRAN symbolic (FTS) and FORTRAN (binary) program (FTB). We would then have the type-structures:

Let us contrast the operations of a text editor, a FORTRAN Compiler, and a user program, with respect to the type-structure. The editor would supply the type S when requesting file name look-up and would accept any file with type FTS, LS, DS, or S. The compiler would supply type FTS and thus exclude type FTB. The user program would ask for files of type DS or some sub-type excluding types LS, and S. Commands to create and modify the type structure are discussed in later sections.

The initial system will have no type structure.

III File Commands

The commands for maintaining a user's file directory (MIB) are invoked in one of two ways. First as commands to the FILE-MAINTENANCE subsystem. This subsystem contains all commands for manipulating the file directory. Secondly, those commands which are frequently used also exist as ordinary "Top-Level" ones. We first describe the commands in the FILE-MAINTENANCE subsystem.

Commands preceeded with + will not be initially implemented.

General

+ >USER user:spec

This command causes the maintenance program to work on the specified directory.

>FINISHED

returns from the maintenance program

+ >CREATE TYPE $name_1$ $name_2$

The first type-name is made a sub-category of the second name.

+ >DELETE TYPE name

This command deletes the specified file type.

Files

For all commands in this category, wherever "file-name" appears, the following convention applies: If either the main-name or type is '*', then the command is repeated for all files which match the rest of the name. The four possibilities are:

name-spec	meaning
* : * or *	all files
* : S	all type S files
SOME:* or SOME	all files SOME
SOME:S	the file SOME:S

>DELETE FILE file-name

If the user executing the command has owner access to the file, it is deleted. If the name refers to a link or access-key, it is deleted also. Processes may not be deleted. If the file-name uses the asterisk convention, the system will ask for confirmation before proceeding.

>RENAME file:$name_1$ file:$name_2$

The name of the object designated by file:$name_1$ is changed to file:$name_2$.

>CREATE LINK file:$name_1$ file:$name_2$

Establishes the first name as a link to (synonym for) the second one.

>DELETE LINK file:name

Deletes the link, not the file for which it stands.

>LIST ALL

Prints the entire MIB contents.

>LIST CO-USERS

Prints the names of all 'friends' and the degree of access to the MIB.

>LIST-LINKS

>LIST-PROCESSES

+ >LIST-KEYS

Print the corresponding entries.

+ >LIST TYPE-STRUCTURE

Prints the complete type structure.

>LIST FILE file-name option

This command lists each designated file-name plus the information specified by the option. If no option is given, only the name is printed.

The options are:

BRIEF which prints the name,

ACCESS which gives the access fields,

LENGTH which prints the current length,

TIME which lists the time the file was last opened and the time it was last modified, or

ENTRY which combines all of the above.

>SET PUBLIC-ACCESS file:name access:code

>SET CO-USER-ACCESS file:name access:code

>SET OWNER-ACCESS file:name access:code

+ >SET KEY-ACCESS file:name access:code key:spec

These commands set the specified access field of the appropriate file(s) to the access-code. The access code may be NO (none), RO (read-only), RW (read-write(, PX (proprietary execution), or OW (owner read/write) or a numeric value. The key-spec may be ## access-key-value, an access-key-name, or a user-spec. Setting KEY-ACCESS to NO causes the key to be removed from the access list.

Co-Users

>CREATE CO-USER user:spec access:code

Causes the user:spec to be added to the friend list if necessary and the access code to be set.

>DELETE CO-USER user:spec

Removes a friend list entry.

Access Keys

+ >GET-KEY file:name$_1$ file:name$_2$

The second file-name, which must be the name of an access-key, is copied to the current MIB as name$_1$.

+ >CREATE KEY file:name

The name is used to create a new access-key.

+ >DELETE KEY file:name

Removes the access key from the MIB.

Top-Level Commands

The following commands will be equivalent to typing:

@FILE-MAINTENANCE

>some command

>FINISHED

The table gives the top-level name and the corresponding sub-command.

Top-Level	Sub-Command
LIST-FILE	LIST FILE
DELETE-FILE	DELETE FILE
RENAME-FILE	RENAME

Following the top-level name the user should type the same parameters required for the corresponding sub-command. For example

```
@LIST-FILE  PROGRAM:*  ENTRY
```

is equivalent to:

```
@FILE-MAINTENANCE
>LIST FILE PROGRAM:*  ENTRY
...
>FINISHED
```

IV The Abbreviation Scan

The main:name can be abbreviated to an arbitrary degree as long as the input string is unambiguous. The precise definition of ambiguity is somewhat complex. However typing the exact name always works, whether or not the name is quoted. We wish to emphasize the point that the abbreviation scan is applied only to main:names, not to user:names, account:names, or types.

A name consists of a number of words separated by hyphens. We first consider the case of single-word names. If a directory contains the names:

LONG

LIST

LISTING,

then "LO," "LIST," and "LISTI" respectively are the shortest abbreviations. In particular "L," "LI," and "LIS" matches a name if the input is identical to the initial characters of the name. Thus "L," "LO," "LON," and "LONG" all match "LONG"; no other strings do. We also distinguish between partial and exact matches.

The abbreviation-scan algorithm for single-word names is this:

1) Scan the directory until a name is found which the input string matches. If no name is matched, return an error.
2) If the input exactly matches the name, return this entry. Otherwise remember this entry.

3) Continue scanning until another match occurs. If no other match occurs, return the entry saved in step 2.

4) If in the previous step we find an exact match, return it. Otherwise the name is ambiguous.

Applying this algorithm to the example above yields the minimal abbreviations listed there.

We now generalize this algorithm to include multi-word names. An input string matches a name if each (partial) word of the input matches the initial words of the name. Note that the name may contain more words than appear in the input string. Thus in the directory:

LIST-FILE

LAB-PROGRAM,

"LI" and "LA" match only one entry. Also note that "L-F" and "L-P" also match a unique entry. The algorithm stated above will work with the generalized definition of match, however it is slightly restrictive. For example to match:

LIST-FILE

LIST-FILES,

would require that the input string be exact, whereas "L-FILE" and "L-FILES" should be adequate abbreviations.

The following algorithm eliminates the restriction stated above. It uses the general definition of match.

1) Scan the directory until a name is found which is matched by the input. If no name is found, return an error. Otherwise save the entry and mark it unambiguous.

2) Continue scanning until another match occurs. Return either the saved entry if it is unambiguous or an error.

3) If two entries (the current and saved ones) are found which are matched by the input, determine which is a more complete match. This decision is made by comparing each word of both entries with the (partial) words of the input. If we find an input word which exactly matches one, but not both, of the corresponding words of the current and saved entries, we choose the entry which has the exact match and return to step 2 (the new choice is unambiguous). Otherwise we mark the saved entry ambiguous and return to step 2.

This algorithm has the following desirable properties:

1) If the input is identical with some entry in a directory, that entry will be returned.

2) Unambiguous abbreviation is allowed.

3) The algorithm can be implemented by examining each entry only once, as opposed to a multiple scan technique. This allows several directories to be "chained" together.