Release 3.0 Notes for the Operating System Reference Manual for the Lisa

These notes reflect changes and additions to the 3.0 release Lisa Operating System software. The notes update the following chapters: Chapter 1, Introduction; Chapter 2, The File System; Chapter 4, Memory Management; and Chapter 5, Exceptions and Events.

Please look over these notes now, before placing them with the corresponding chapters in your binder for easy reference.

030-1110

Chapter 1
Introduction

Using the SYSCALL Unit

If a Pascal program contains Operating System user-interface procedure calls, then the program's USES clause must specify the SYSCALL unit, contained in the SysCall.Obj file:

```
Program MyProg;
USES {$U SYSCALL.OBJ} SysCall;
```

Chapter 2
The File System

New Hierarchical File System

Each mounted disk volume now has a hierarchically arranged directory structure. The root directory of a volume is always present, and subdirectories may be created to contain collections of files that are logically related.

Path Names (See Section 2.1)

A particular file or directory is specified to the file system with a *path name*. A path name is a sequence of directory names, separated by dashes (-), ending in a file or directory name. For example, the path name

-lower-memos-conference.text

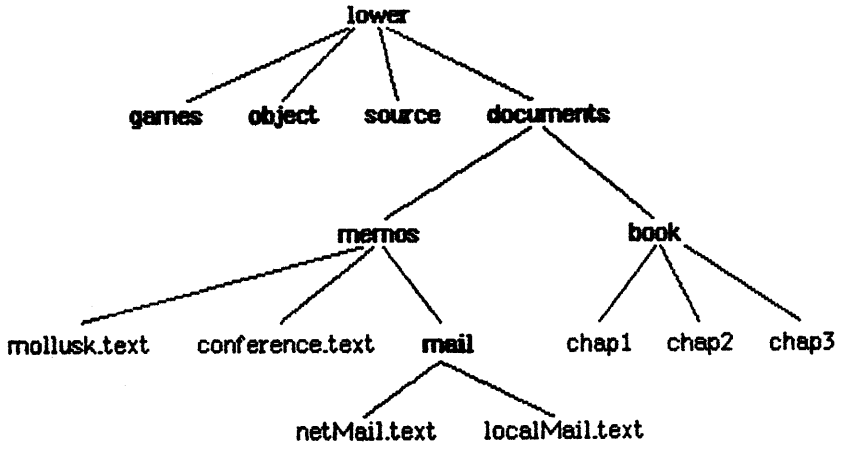
specifies that the root directory of the disk volume *lower* be searched for the directory *memos*, and then *memos* be searched for the file *conference.text*. File and directory names are limited to 32 characters in length, and are truncated to 32 characters if too long.

The Working Directory (See Section 2.2)

A working directory is associated with each process in the system. When a process is created, its working directory is the root directory of the boot disk volume. A process may reassign its working directory through the **SET_WORKING_DIR** call. The **GET_WORKING_DIR** call returns the path name of the working directory in a printable string. A path name submitted to the file system by a process may be specified relative to the process's working directory. This is done by omitting the initial dash from the path name. Suppose that the working directory is *-lower-documents-memos* in the directory hierarchy shown below. The path name *mail-netMail.text* specifies that the working directory be searched for the directory *mail*, and then *mail* be searched for the file *netMail.text*. The path name *conference.text* specifies that the working directory be searched for the file *conference.text*.

The plus delimiter (+) may be substituted for the dash within a path name to indicate that the next directory in the path name is the parent of the preceding directory. The plus delimiter is typically used to trace a path by moving upward in the directory tree relative to the working directory. Again suppose that the working directory is *-lower-documents-memos*. The path name *+documents-book-chap3* specifies that the parent directory of the working directory be searched for the directory *book*, and then *book* be searched for the file *chap3*. The path name *+documents+lower-games* traces a path up through directory *documents* to the root directory of disk volume *lower* and then down to find directory *games*. Since the parent directory of

any given file or directory is unique, the name following a plus delimiter within a path name may be omitted. For example, the path name *++games* is equivalent to *+documents+lower-games*, and the path name *+* is equivalent to *+documents*.



Directory Tree

Pipes (See Section 2.9)

The pipe facility has been removed. **MAKE_PIPE** has been deleted, and any attempt to **OPEN** an old pipe object will return an error number 948. All the inter-process communication (IPC) features provided by pipes are also provided by event channels.

MAKE_CATALOG File System Call

```
MAKE_CATALOG (var ecode : integer;
              var path : pathname;
                  label_size : integer)

ecode:      Error indicator
path:       Name of the new catalog
label_size: Number of bytes for the catalog's label
```

MAKE_CATALOG creates a catalog (also called a directory) with the specified pathname. **label_size** specifies the initial size in bytes of the label. It must be less than or equal to 128 bytes. The label can grow to contain up to 128 bytes no matter what its initial size. Any error indication is returned in **ecode**.

QUICK-LOOKUP File System Call

```
QUICK_LOOKUP (var ecode : integer;
              var path : pathname;
              var InfoRec : Q_Info)
```

ecode: Error indicator
path: Name of the object to lookup
InfoRec: Information returned about the object

QUICK_LOOKUP returns information about a file or directory. **QUICK_LOOKUP** is significantly faster than **LOOKUP** (about five times), but returns a subset of the information available through **LOOKUP**. **QUICK_LOOKUP** is not applicable to a disk volume or device, only to files or directories. The definition of the **Q_Info** record is shown below; note that many of the fields are not defined when **QUICK_LOOKUP** is applied to a directory.

```
Q_Info = RECORD
           name          : e_name;
           etype         : entrytype;
           DTC           : longint;
           DTM           : longint;
           size          : longint;
           psize         : longint;
           fs_overhead   : integer;
           master        : boolean;
           protected     : boolean;
           safety        : boolean;
           left_open     : boolean;
           scavenged     : boolean;
           closed_by_OS  : boolean;
           nreaders      : integer;
           nwriters      : integer;
           level         : integer;
         END;
```

name: Name of the file or directory.
etype: Type of object (either fileentry or catentry).
DTC: Date/time the file or directory was created.
DTM: Date/time the file was last modified.
size: Number of data bytes in the file (LEOF).
psize: Physical size of the file in bytes.
fs_overhead: Number of disk pages used by the file system to store control information about this file.
master: Flag set if this file is a master.

protected:	Flag set if this file is protected (refers to software theft protection, not password protection).
safety:	Flag set if the file safety switch is on.
left_open:	Flag set if this file was left open during a system crash.
scavenged:	Flag set by the Scavenger if this file has been partially or totally rebuilt.
closed_by_OS:	Flag set if this file was last closed by the Operating System.
nreaders:	Number of processes with this file open for reading.
nwriters:	Number of processes with this file open for writing.
level:	Level of the file or directory within the directory tree. This field has valid contents only when the Q_Info record is returned by LOOKUP_NEXT_ENTRY.

Chapter 4
Memory Management

Memory-Resident Data Segments (See Section 4.1)

There is a limitation on the usage of memory-resident data segments. A data segment created using **Make_Dataseg** with 0 disk space cannot have its disk size subsequently increased with a **Size_Dataseg** call. If you want to be able to assign disk space to a memory-resident data segment, create the segment initially with some disk space (e.g., one page), then reduce the disk size immediately to 0 using **Size_Dataseg**. Later, you can increase the disk size of the memory-resident segment using **Size_Dataseg**.

Chapter 5
Exceptions and Events

Event Channels (See Section 5.5)

Timed event channels have been removed.

Event channels are now memory-based rather than disk-based. This means that event channels are not preserved across system activations. If the system is shut down, all event channels are deleted. Any data that must be preserved should be read from an event channel and stored in a file until needed the next time the system is booted.

In the example on page 5-7, the boolean **receiver** is mistakenly set to TRUE and then FALSE--it should be FALSE then TRUE.

SET_LOCAL_TIME_DIFF (See Section 5.9.3)

The SET_LOCAL_TIME_DIFF clock system call has been removed.