E1 MOS

L1 MOS

Introduction to MOS

olivetti

PUBLICATION ISSUED BY:

Ing. C. Olivetti & C., S.p.A.
Direzione Documentazione
77, via Jervis - 10015 Ivrea (Italy)

PREFACE

This manual is intended for users who have not used MOS (the L1 line Operating System) before, but who are familiar with computer systems.

It gives enough information about MOS for the new user to begin to consult other documents in order to use the system.

SUMMARY

Introduction to MOS contains the following sections:

- A general overview of the system, which covers possible configurations and application areas.
- Chapters on logging in and out, the SHELL interface, file and volume handling, work station handling, and remote services in distributed configurations (part 1 - Using the System).
- Chapters on the programming languages available under MOS, support packages, software environments and general programming tools (part 2 - Application Programming).
- Chapters concerning system programming and system generation, installation and maintenance (part 3 - Tools for System Specialists).

REFERENCES

Read first ...

L1 MOS System Summary - Code 3986250 E

For further information, read ...

Programming Languages - Features and Functions
Code 4004210 Z (Vol. 2)

DMS - Features and Functions - Code 4002610 Q (Vol. 2)

VISA - Features and Functions - Code 4002630 J (Vol. 2)

CSS - Communication Subsystem Features and Functions
Code 4000830 S (Vol. 2)

MTS - Features and Functions - Code 4003390 M (Vol. 2)

BEAM - Features and Functions - Code 4003160 F (Vol. 2)

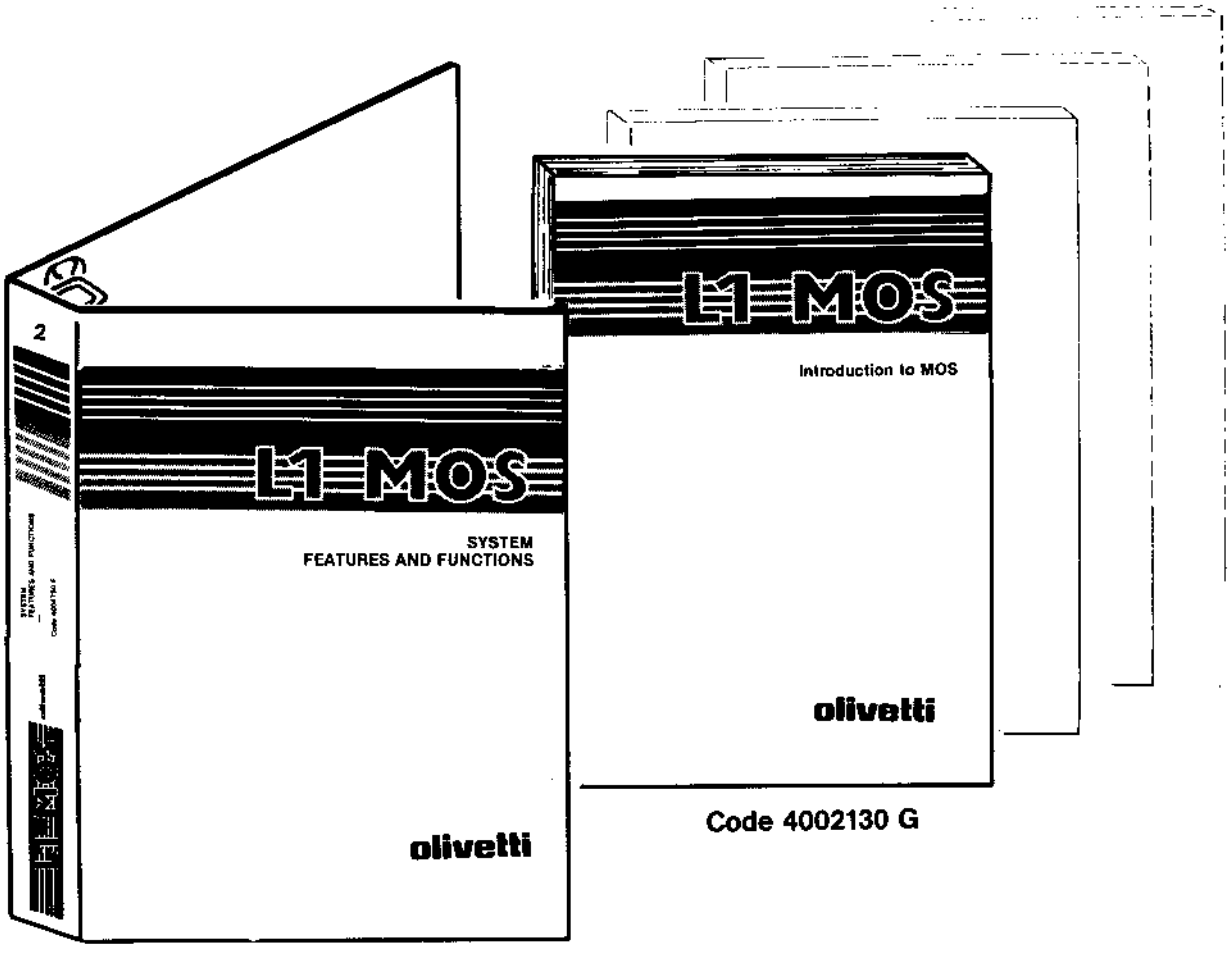
Introduction to ONE - Code 4008530 A (Vol. 2)

MOS - Programmer Guide - Code 4002570 L (Vol. 6G)

Glossary/Glossario - Code 4002140 H (Vol. 1)

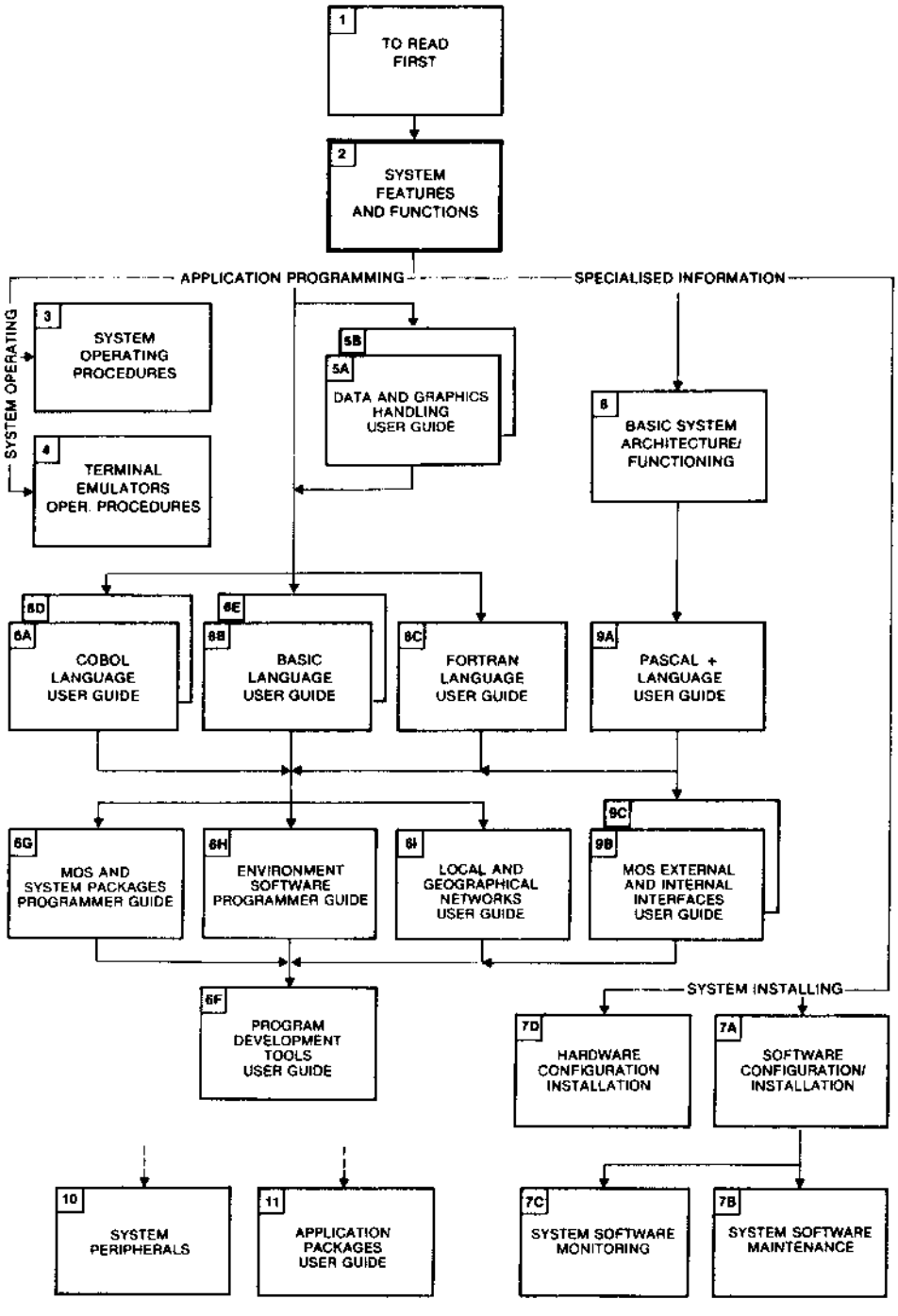
FOURTH EDITION: October, 1985 - Release 5.0/5.1

FIFTH EDITION: June, 1987 - Release 5.2



Code 4004750 F

Code 4002130 G


1
TO READ FIRST
2
SYSTEM FEATURES AND FUNCTIONS
APPLICATION PROGRAMMING
SPECIALISED INFORMATION
SYSTEM OPERATING
3
SYSTEM OPERATING PROCEDURES
4
TERMINAL EMULATORS OPER. PROCEDURES
5B
5A
DATA AND GRAPHICS HANDLING USER GUIDE
8
BASIC SYSTEM ARCHITECTURE/ FUNCTIONING
6D
6A
COBOL LANGUAGE USER GUIDE
6E
6B
BASIC LANGUAGE USER GUIDE
6C
FORTRAN LANGUAGE USER GUIDE
9A
PASCAL + LANGUAGE USER GUIDE
6G
MOS AND SYSTEM PACKAGES PROGRAMMER GUIDE
6H
ENVIRONMENT SOFTWARE PROGRAMMER GUIDE
6I
LOCAL AND GEOGRAPHICAL NETWORKS USER GUIDE
9C
9B
MOS EXTERNAL AND INTERNAL INTERFACES USER GUIDE
6F
PROGRAM DEVELOPMENT TOOLS USER GUIDE
SYSTEM INSTALLING
7D
HARDWARE CONFIGURATION INSTALLATION
7A
SOFTWARE CONFIGURATION/ INSTALLATION
10
SYSTEM PERIPHERALS
11
APPLICATION PACKAGES USER GUIDE
7C
SYSTEM SOFTWARE MONITORING
7B
SYSTEM SOFTWARE MAINTENANCE

CONTENTS

1. OVERVIEW

MOS (Multifunctional Operating System) is a general-purpose, multi-user, multi-tasking, and multi-processor operating system. It has been designed to support a wide range of services and facilities for different application environments.

From the user's point of view the most relevant feature is the MOS Command Language (MCL), which provides a simple and easy-to-use tool for the control and execution of user jobs and for calling operating system services.

For application programming, the system provides high-level programming languages such as COBOL, BASIC, and FORTRAN; PASCAL+ may be used for system programming. A COBOL ICE program generator (CRECOS) is also supplied for an easy preparation of COBOL source programs.

MOS has a powerful, user-oriented File System, whose files have the most common types of organisation.

MOS handles standard peripherals, such as terminals, disks, printers, and special peripherals required by particular applications. It can also be used in multi-user mode as it is capable of guaranteeing concurrent access to the various system resources such as files and printers, and handling job queues.

MOS is also a distributed operating system: it guarantees to all its application environments equal visibility of the various resources, even when they are installed on different processors connected together. The same application program can be used on stand-alone and distributed configurations.

POSSIBLE CONFIGURATIONS

MOS can operate in any meaningful combination of the following communication topologies:

- Stand-alone, in single or multi-user mode
- As part of a cluster (in the case of a local network)
- An intelligent terminal for a host computer, with possible initialisation and/or shutdown by host
- A node in a local or geographical network.

Two operating modes are available under MOS for program execution in time-sharing: interactive (most commonly used) and batch.

In these two modes the central unit activity is divided in time-slices, and MOS assigns an appropriate amount of CPU time to the various processes making requests. This means that interactive processes do not suffer in favour of batch processes that require more CPU time.

Activities can, however, be executed in real-time, which means that high-priority processes are executed immediately. Only when these are finished will MOS newly assign the central unit to time-sharing processes that have lower priority.

Multitasking

MOS can handle several processes at the same time, controlling them simultaneously using several windows. The windows are activated on the same screen. The user can move from one window to another by pressing a key.

Multi-processor

MOS can handle the M60 system in the multi-processor version (bi-triprocessor). The multi-processor system has symmetrical architecture: each activity is executed in the first free CPU. The most basic components of the operating system are available in two versions: one of these is specifically for mono-processor systems, and the other is for multi-processor handling and can be used on all L1/MOS models.

MULTIFUNCTIONAL CHARACTERISTICS

The MOS operating system has been designed to support various applications typical of different environments; the following is an indication of the possible problem areas in which MOS may be used:

- Data Processing (for example, data entry, business applications)
- Distributed Data Processing
- Banking (using distributed techniques and special peripherals)
- Computer-Aided Design/Manufacturing (CAD/CAM)
- Science and technology (for example, interactive plotting, experiment control, up-to-date graphics)
- Engineering and process control
- Public and private networks
- Office automation for local and regional networks

The different application areas are managed by MOS in conjunction with system packages and specialised software, known as environmental software, which aims to provide the user with a powerful and application-oriented system environment.

Some of the system packages offered are:

- a set of powerful graphic software packages, used to write application programs for two-dimensional graphics, ranging from the simple to the most highly interactive, and to display mathematical, physical and economic functions for business, science and industry

- a screen-layout handling package, including an interactive screen and report layout generator and a mask handler, which controls the input of data and formatting of output for the printer (VISA).

Examples of software environments are:

- the MTS transaction handler (Modular Transactional Support), which supports distributed transactional applications made up of interactive programs (IE, Interactive Environment), which handle the system/operator dialogue, and shared programs (SE, Server Environment), which provide services and are called by the interactive programs.

- BEAM, which handles business environment resources and users

- terminal emulators

- a data management system for the creation and updating of a data base and inquiry about its contents and for report production (DMS)

- office automation facilities, such as word processing, table handling, form processing, office procedures, dossier handling, agenda and electronic mail facilities (for further information, see the manual OWS-2 Office Automation with L1 MOS, Code 4001290 C

- the ONE environment (Open Network Environment), which connects OLIVETTI systems in geographical networks (X25 or SNA) or local networks (Ethernet or Olinet), offering users a data exchange and file transfer service in line with ISO network architecture.

- OLILAN (OLIVETTI Local Area Network), which enables L1 systems and OLIVETTI Personal Computers to be connected in local networks, offering PC users a complete range of services (sharing of common peripherals within the network, execution of remote programs, etc.)

- an emulation environment (ESE - Extended System Environment) for OLIVETTI P6066 systems, which supports an extended BASIC language and guarantees maximum compatibility with the preceding scientific/technical OLIVETTI systems (for further information, see the manual ESE Reference Manual, Code 3930860 R).

The multifunctionality of the MOS operating system means that all the environments and languages (provided they are supported by the present hardware) can be used simultaneously by different work stations.

MODULAR CHARACTERISTICS

The MOS operating system has been designed to be highly modular so that it can be configured to meet the user's needs. Individual systems may include just those modules that relate to the user's particular application area, and exclude other modules that are not necessary.

CONFIGURATION EXAMPLES

The examples below indicate some possible MOS configurations, starting with the most basic system with little memory and going on to more complex systems that require more control and auxiliary memory.

1. Single Terminal, minimal run-only (based only on diskette) configuration for commercial applications using COBOL.

   This configuration enables the user to run, in an interactive mode, programs that have been developed on another system.

   In addition to the basic components (kernel, system peripheral drivers, process and memory manager) the configuration includes:

   - the File System, which supports byte-stream and positional files
   - keyed (mono-key and multi-key) file access
   - COBOL run-time support
   - the Sort/Merge utility (optional).

2. Multiple terminal, run-only, on-line configuration for commercial applications using COBOL.

   This configuration allows two or more users to run simultaneously, in an interactive mode, programs that have been developed on another system.

   In addition to the support for those features described in 1., the system also includes:

   - the Communication Subsystem for handling line communication to Host with IBM 3270 protocol
   - the general purpose Shell software environment and relative interactive MOS command language (MCL), without the program preparation commands
   - user login-logout (including password) support, requiring a system administrator
   - file protection support
   - the VISA package for handling video formats

- a spooler for handling the system printer

- batch job monitor

- in the case of magnetic tape configurations, for file back-up the utilities for volume dump/restore to and from hard disk.

3. Multiple terminal, program development configuration for commercial applications using COBOL.

   This configuration supports program preparation on several terminals while applications are being executed on others.

   The following facilities are thus available:

   - program and text editing

   - program compilation

   - data definition and file handling

   - definition of the screen format.

   In addition to the support for those features described in 2., the system also includes:

   - the Editor

   - the COBOL compiler

   - the CRECOS program generator (optional)

   - the Sort/Merge Language Analyser, which allows sort/merge procedures preparation using the appropriate command language

   - Shell commands for program preparation and file handling.

4. Multiple terminal, run-only, multifunctional, program development configuration using BASIC and FORTRAN, and the ESE environment (OLIVETTI P6066 Systems emulation).

   This configuration is typical of the scientific/technical environment. It allows the user to write or execute BASIC programs interactively, or to execute FORTRAN programs, for business and scientific/technical applications.

   Programs written in different languages can exchange information using files accessed at different times. These files must be homogeneous to the two languages.

   In the ESE environment, which can be installed on several terminals, it is possible to:

   - write, debug and execute BASIC-ESE programs

- process graphics and texts
- execute immediate calculations
- handle I/O operations with external serial peripherals.

This environment cannot exchange information with programs written in the other available languages.

This configuration includes the support for the features listed for 2., with the addition of:

- support for handling black/white and colour graphic screens, with hard copy features on graphic printers, and scientific keyboard handling
- support for handling the RS232/CL interface, which allows any input/output serial peripheral to be connected (for example, a plotter)
- the DMS (Data Management System), which allows the creation and modification of files, file inquiry, file copying, printing, via a particular command language (DML)
- the access method to keyed files (optional)
- support for handling the multi-user environment.

5. Configuration for program development multi-keyboard, multi-functional with COBOL and BASIC languages.

   This configuration allows programs that use graphics features on (say) two terminals, and the administrative programs on the other terminals, to be simultaneously prepared and executed.

   Both COBOL and BASIC, giving the possibility of handling shared files, allow programs which are written in different languages and executed from different work stations to access files concurrently, provided that their types are homogeneous.

   In comparison with example 3., this configuration enables (via BASIC):

   - handling of graphic features
   - handling the RS232 interface.

6. Cluster configuration ("run-only", multi-terminal, on-line configuration in COBOL) for a transactional application. This configuration has the following:

   a) Master configuration: as well as the components described in 2., this configuration includes:

      . the primary protocol, in the Communications Subsystem, for handling the "internal" communication line (MOS to MOS)

- support software to remote requests from the satellites

- the MTS transaction handler, including the Transactional and Interactive Environments (TE and IE)

b) Satellite configuration: as well as the components described in 1., this configuration includes:

- the VISA package for handling video formats

- the MTS transaction handler's Interactive Environment

- the support software for access requests to the master's services

- the secondary "internal" line protocol

- optionally, the interface for remote access to the "external" line.

7. Configuration for operating system generation and maintenance. This configuration, which is typical of software distribution centres, includes:

- a basic extended software with maximum functions, including all the drivers necessary for controlling a large number of magnetic peripherals

- field support utilities (FSU), which allow MOS and its various packages and application environments to be configured and application environments to be configured and generated

- installation and maintenance commands, which allow the configured software to be installed, and which help OLIVETTI maintenance specialists when installing systems

- the PASCAL+ language with compiler, linker, debugger and the other tools for program development on the system itself.

PART 1 - USING THE SYSTEM

INTRODUCTION TO PART 1

The first part of this manual gives information on how to use the system.

The description includes the operating procedure to be followed when initialising the system, selecting the work environment, and carrying out activities in Shell environment.

The system command language, use of the File System, and the system drivers available are then illustrated.

The last chapter in this part of the manual illustrates how MOS is used in a distributed configuration.

2. OPERATING PROCEDURES

This chapter includes the most important aspects of how to operate the system:

- Initialisation
- Entering the system
- Work environment selection
- Job execution
- sending messages to the operator
- Shutdown

SYSTEM INITIALISATION

The system is automatically initialised when switched on, thus activating the:

- autodiagnostics
- loading of the operating system.

The operating system is loaded from a diskette, hard disk, or tape (SCT or MTU), depending on the system's hardware configuration. Selection of any of these on systems supplied with hard disk is made by using the appropriate switch (IPL switch, see Fig. 2-1). When using tape as the initialisation support, the system is first installed on hard disk and then loaded from there into memory. Initialisation is subsequently carried out directly from hard disk when the machine has been reset.

System Date

On the M6x/M70 and M3x/M4x (if suitably configured) the system date and time are remembered while the system is switched off.

Unattended Mode

On the M3x/M4x (with the Automatic Start Device - ASD 3384), or on the M6x and M70, initialisation operations can be carried out on request, using the communication line from a remote system. If the system has been suitably set up, on initialisation it automatically executes the preset activities without operator intervention.

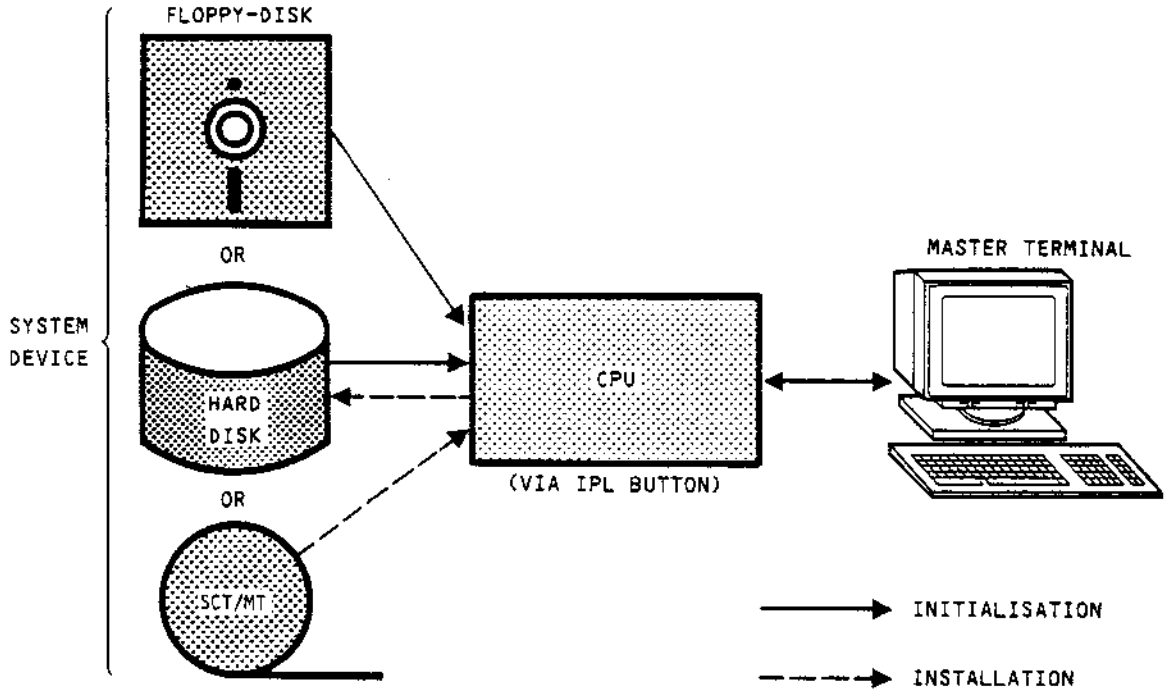


Fig. 2-1 System Initialisation

SELECTING THE ACTIVITY

When system initialisation, or IPL (Initial Program Loading), has been correctly terminated, a system process (Grandpa) is automatically activated. This process starts the application activities on the various system terminals, and also starts the background activities. The term "activity" means:

- an application program
- a system program (for example a utility)
- a package that, when run, determines a specific application environment in which application or system programs can be executed
- a non-interactive program (for example, the spooling or batch job monitors).

The application environments that can be activated by Grandpa are:

- Shell
- Transaction Handler MTS
- Business Environment Application Monitor BEAM
- COBOL ICE
- Interpreted BASIC
- Data Management System DMS
- ESE
- Terminal Emulators
- Local Area Network LAN
- Open Network Environment ONE
- SNA T5 node

Every application activity is executed with a priority that depends on the activation mode specified in the Grandpa configuration file. This mode can vary according to the type of program to be executed (interactive, non-interactive, network server, user package, ...). Further details about this topic can be found in the "Activating the Programs and User Subsystem" of the MOS Programmer Guide manual.

USER ENVIRONMENT CONFIGURABILITY

Grandpa sets up the specific program, environment, or set of environments, for each terminal according to the information contained in its configuration file. In the last case, after the login phase (see the "User Management" section) a menu is displayed and the operator can select the desired activity.

Grandpa's configuration file allows the functions of each system terminal (uniquely identified, see "Devices" Section, Chapter 4) to be defined, restricting the accessible functions of one terminal and expanding those of another, according to the requirements of the application.

For example, one of the terminals can be defined as the 'master', enabling it to carry out certain system functions, such as initialising the system clock, activating the shutdown procedures, and issuing warning messages. An environment for preparing programs in one of the available languages can be associated with another terminal, one or more application activities can be associated with yet another, and so on for all the terminals. The terminals whose screens are to be divided into windows can be specified, and a user can be assigned a specific activity on a specific terminal (thus avoiding the necessity to log on to that terminal), and so on.

The configurability of Grandpa means that a large number of possible software configurations are available: for example, the Interpreted BASIC

environment can be activated even in a system where the Shell environment is not present.

To allow operations in this type of configuration, Grandpa allows, among the activities that can be selected by the master terminal, the logical connection and disconnection of private and system volumes, thus permitting the use of different magnetic supports.

After installation, the user can modify Grandpa's configuration file in order to change the distribution of the activities among the various terminals.

The operations of the system's basic application environment, the Shell environment, are described later.

Further details on the other environments that can be activated by Grandpa are given in the relative features and functions manuals.

ACTIVITY EXECUTION

In an MOS environment an executable activity means a program, comprising code and data, that is loaded into memory from suitably formatted files (called l-modules).

Execution of a series of program instructions forms a process.

One or more processes form a family, which is the entity that is assigned control of the processing unit.

Families compete among themselves for control of the CPU, which is assigned to them according to execution time used ("time sharing") and their relative priorities. When CPU control is taken from one family and given to another, the system stores a set of information regarding the interrupted family. When the latter regains control, execution is taken up from the exact point at which it was interrupted.

Further details on this subject are given in the Chapter "Segments, Families, and Processes" in the MOS Programmer Guide.

THE MASTER TERMINAL

It is possible, during the configuration phase, to define any one of the connected terminals as the 'master'.

At system start up, the user login prompt, if present, is always displayed on the master terminal (which must be switched on). However, only privileged users (those belonging to the group entitled to perform certain system functions - usually ROOT) can perform the login operation. Thereafter, any privileged user can perform any of the operations of the user environment configuration phase (for example, inputting the system time, changing the system disk, activating initial programs). Having completed this phase, the login prompt is displayed on all the terminals, including the master.

From this moment on, the terminals can be used to perform any of the available activities. The system manager can use the master terminal to exchange messages with the other users. In fact, the user of a terminal in the Shell application environment can also send messages to the system manager (CALLMWS command).

The system manager is notified by the appearance of a warning message in the lower window (twenty-fifth line) of the master terminal when a message arrives. This, however, assumes that the split window option for this terminal was chosen during the user environment configuration phase. When the system manager wishes, he can read the contents of any message (MTA command), which is displayed in the upper window. The terminal used need not be the master.

Local Network Configuration

When several systems are interconnected on a local network, each one can contain a master. A "global master" can be defined for all the network users.

USER MANAGEMENT

In multiuser systems the users are identified by a login name and, possibly, by a password.

This name is assigned by the System Administrator (whose name is ROOT) who is the only person authorised to create new users (USERMAN command).

Users can only modify their own password (SETPASSW command).

The users are subdivided into groups, one of which is composed of privileged users (for example, ROOT). They can execute certain system functions during the initialisation phase, and can request system shutdown.

USER LOGIN

After the IPL phase, the user login prompt is displayed on each terminal if a login is required. Having recognised the user-login entered, the system prompts for a password. When the password has been accepted, the terminal is available for use.

User .MESG File

Each user has a .MESG file under his directory. (See Chapter 4.) This file has several purposes:

- It receives and stores messages sent out by a user's batch job.
- It receives and stores messages sent by other users when the receiver is logged out.

- It receives and stores long messages not coming from MCL procedures.

If the .MESG file is not empty, when the user logs in a message appears on the screen to tell him so.

The user may later access his .MESG file to read the messages that have been stored in it, by printing them out or displaying them.

USER LOGOUT

To terminate an activity, the LOGOUT command is entered: this will disconnect the user from the system.

The main menu is then displayed on the screen (or the associated activity is reactivated, if the work station is not multifunctional).

The user can:

- start any of the available activities
- select LOGOUT and end the work session.

If the second of these two options is chosen, the user login prompt is displayed on the screen.

SHELL ENVIRONMENT ACTIVITIES

The user can start a work session after logging in and entering under Shell.

SHELL WORK SESSION

During a Shell session the user may:

- execute jobs (MCL commands, MCL procedures or programs)
- enter a sub-environment for the preparation and execution of BASIC, COBOL ICE or DMS programs.

THE SCREEN

In Shell environments the screen is divided into two windows (see the section "User Environment Configurability"). The first, called the "synchronous window", formed by all the screen lines except the last, is the standard output for the current work session.

The second, called the "asynchronous window" or "system line" is formed by the lowest line of the screen and remains active in sub-environments. It remains active also in sub-environments activated by Shell.

To pass from one window to another, simply press the **CH.WIND** key; to activate the system line, press **CNTRL K.**

STANDARD I/O

For the purpose of I/O operations all programs, commands or procedures executed during a Shell work session are automatically connected to the work station from which they were entered (standard input/output). This is also true for printing operations carried out on the printer associated at system configuration time with the work station at which the command was entered. In the absence of this printer the text is usually automatically displayed on screen without user intervention.

When a job is run, the user can explicitly request modifications to the standard input/output.

The output can be directed onto a file, a system printer, another work station (terminal) or onto the spooler.

The input can be redirected from a file or from another work station.

JOB EXECUTION

In the Shell environment 'job' means a command, an MCL procedure (see Chapter 3), or an executable program. Jobs may be executed in interactive or batch mode.

Interactive Jobs

The user has only to enter the command or the name of the program or procedure (together with the parameters, if any) in order to execute the job in interactive mode.

Having done this, Shell will start executing the job. Other jobs can be nested within a program or a procedure.

When this interactive job execution is terminated, the user may request Shell to execute other commands, procedures, or programs.

An MCL input-line (see Chapter 3), entered on the system line (that is, the last screen line) will automatically suspend the running of the current job and the requested command, procedure, or program will be executed. Following this, the suspended job will be automatically reactivated.

Control of Shell may be returned to the user by entering the KILL or SUSPEND commands on the system line. The suspended job may only be reactivated by the RESUME command. A job that has been deleted by the KILL command cannot be reactivated.

Batch Jobs

Jobs are executed in batch mode by entering the BM command before the command, procedure or program name.

The job is placed in an appropriate queue and executed independently of

the activities currently being executed on the terminal from which it has been run. The operator retains control of the screen when executing jobs in batch mode.

Each batch job is identified by a unique number. Its status and position in the queue may be displayed on the screen, and the queue may be displayed on the screen and modified with an appropriate command (BATCH), which handles all operations on the queue.

Spooler

Files to be printed may be submitted to the spooler using a parameter in the LPR command. The print job is placed in a queue to be printed on the selected system printer. This allows the operator to continue using his terminal.

Each print job is identified with a unique number. Its status and priority may be displayed and modified using the SPOOL command. The system administrator must allocate queues to any available system printers and may control the printing of the queues.

OPERATOR MESSAGES

The Shell environment can interact with the operator through:

- asynchronous messages, which may be generated by a batch job, or, in cases of malfunction, by the operating system
- synchronous messages, which are generated by the current interactive application.

Asynchronous messages are displayed in the "asynchronous window" that is generated by the Shell and known as the "system line" (the bottom screen line).

Synchronous messages emitted by a line-oriented interactive activity are displayed on the line immediately after the last line entered by the user.

SYSTEM SHUTDOWN

The system shutdown (switching off) phase involves certain preliminary operations.

The request for system shutdown can be made:

- in interactive mode
- by a program.

An interactive shutdown request may only be made by a privileged user (that is, a member of the group permitted to execute system functions - usually ROOT), who selects the SHUTDOWN activity on one of the terminals he is using.

A program that makes a system shutdown request may only be run by a privileged user. The system is shut down if, on completion of execution, the program returns a particular code (a facility particularly useful for systems initiated in unattended mode). A different return code permits the system shutdown to be followed by re-initialisation (normally for Software Distribution; see Chapter 2, "Networks").

On completion of the request, a message appears on all the terminals connected to the system to warn users to close their sessions.

When the defined time limit has expired, the shutdown procedure is activated and the power can be switched off.

3. THE MOS COMMAND LANGUAGE

MOS Command Language (MCL) is interpreted by Shell, and provides an easy-to-use interface to MOS.

In order to use MCL, the user must log in to the Shell environment. Having done this he may:

- execute standard MCL commands
- enter a line of MCL statements to be processed interactively by Shell
- create or run user procedures composed of MCL statements
- execute programs
- activate the BASIC, COBOL, or DMS program preparation and execution environments.

MCL COMMANDS

This section presents the commands that may be used in the Shell environment. These commands are grouped by function as follows:

- User and password management commands
- System clock management commands
- Directory management commands
- Volume and disk management commands
- File management commands
- Family management commands
- Process management commands
- Print commands
- Display commands
- Message transmission commands
- Queue management commands
- Diskette conversion commands

- Save/Restore commands
- Data integrity commands
- Global resource management commands in the distributed environment

The commands followed by an asterisk are only available for the system manager.

See also Chapter 4 (Handling Files and Volumes) for a complete definition of the File System objects (volume, directories, files) that may be manipulated by some of these commands.

The HELP command will display a concise description of the most frequently used commands.

Note:

Not all the MCL commands described below are included in all the system configurations. Run-only systems do not have any program preparation commands. In any case, any command can be added or removed in order to personalise the configuration.

See the SHELL, Operating Commands, Reference Manual for a complete description of these commands.

USER AND PASSWORD MANAGEMENT COMMANDS

FINGER	Displays information about the users.
WHO	Displays a list of the names of all the users currently logged in.
USERMAN	Handles the creation and removal of users. (*)
SETPASSW	Permits a user to define, change, or delete his own password.

SYSTEM CLOCK MANAGEMENT COMMANDS

SETDATE	Modifies the system date and time. (*)
SHDATE	Displays the system date and time.

DIRECTORY MANAGEMENT COMMANDS

MKDIR	Creates a directory.
CLEARDIR	Deletes the contents of a directory.
SHDIR	Displays the contents of a directory.
CHTYPE	Converts a normal directory into a program-directory, and vice-versa.

VOLUME AND DISK MANAGEMENT COMMANDS

MKVOL	Creates a volume within another volume or initialises a disk (HD/FD) creating the principal volume.
MNT	Logically connects a volume to the system.
UNMNT	Logically disconnects a volume from the system.
SHLABEL	Displays an unmounted disk label (FD/HD).
MKENV	Initialises a floppy disk. (*)
VCOMPACT	Optimises disk space occupation. (*)

FILE COMMANDS

MKBST	Creates a byte-stream file.
MKPOS	Creates a positional file.
MKKEYED	Creates a keyed file.
MKINDEX	Creates a secondary index for a keyed file.
MKALIAS	Creates an alias file.
SHALIAS	Displays the contents of one or more alias files in ASCII.
DELINDEX	Deletes an index of a keyed file.
REMOVE	Deletes files, empty directories, or volumes.
RP	Displays the physical description of a file.
COPY/CPTREE	Copies files, directories, or volumes.
MORE	Displays the entire contents of a sequential or byte-stream file in ASCII.
LIST	Displays the whole or partial contents of a file in ASCII or hexadecimal code.

SETINFO	Defines or modifies the attributes of a file.
PRY	Displays the characteristics of a file, directory, or volume.
RCODE	Displays the release level of a loadable system module (l-module).
CONVERT	Converts a file from one type to another. Types may be byte-stream, positional or keyed.
RENAME	Renames a file, directory, or volume.
COMPACT	Optimises the memory occupation of a file.
CMPK	Compacts a keyed file or positional packed file.
DIFF	Compares two byte-stream files, or two physical devices, byte by byte.
EXIST	Verifies the existence of a file.
WHEREIS	Looks for files, directories, or volumes.
WSTAT	Displays the physical description and the logical connections of a file.
LS	Displays information about files, directories, or volumes.
SETBUF	Reserves a private buffer pool for I/O operations on a particular file.

COMMANDS FOR HANDLING FAMILIES

KILLFAM	Deactivates one or more families (including those in the background). (*)
RESTART	Causes a shutdown followed by a new system initialisation. (*)
SHFAM	Displays information on the families active in the system (including those in the background).

COMMANDS FOR HANDLING PROCESSES

KILL	Deletes an interactive job from the system.
SUSPEND	Temporarily suspends the execution of an interactive job.
RESUME	Resumes the execution of a job halted by SUSPEND.

PRINT COMMANDS

LPR	Prints the contents of one or more sequential or byte-stream files (of ASCII characters) or sends the contents to the spooler.

DISPLAY COMMANDS

SHOW	Writes the specified string on the terminal screen or on any other system device (printer, disk file).
CLEAR	Clears all characters and other visual attributes present on the terminal screen.
SHTERM	Displays the terminal screen and keyboard characteristics.
SETTERM	Defines the screen format (how many characters can be displayed).

MESSAGE TRANSMISSION COMMANDS

WRTUSER	Sends a message to a specific user.
WRTALL	Sends a message to all the users logged on to the system.
CALLMWS	Sends a message or data to the system administrator at the master terminal, and may wait for a reply.
MTA	Receives and handles messages for the system administrator. (*)

QUEUE HANDLING COMMANDS

SPOOL	Handles the spooler queue.
LPQ	Displays the status of a printer, or of its queue.
BATCH	Handles the Batch monitor queue.
BMQ	Displays the state of the batch job queue.

DISKETTE CONVERSION COMMANDS

FCU	Converts floppy disks from standard OLIVETTI to non-standard OLIVETTI and vice versa (see the section "Utilities").
FCUTAB	Creates a user-defined conversion table to be used with the FCU command.
TCU	Transfers files from an L1 system to another system, with the applicable conversion, by means of a magnetic tape.

SAVE/RESTORE COMMANDS

FILETAR	Places/removes files, directories, or volumes on/from tape archives.
FLD	Copies a volume from hard disk to one or more diskettes or vice versa. (*)
VOLSR	Copies one or more volumes from hard disk to SCT and vice versa. (*)
FLDUMP	Copies a volume from hard disk to one or more diskettes or vice versa. (*)

DATA INTEGRITY COMMANDS

COMMIT	Activates a program under the Commit Manager's control.
CLOSECOMM	Suspends the Commit Manager's activities.
OPENCOMM	Reactivates the Commit Manager.
DISPJOUC	Displays the JOUCMAN file's contents.
RECOVERY	Recovers a damaged file.

GLOBAL RESOURCE MANAGEMENT COMMANDS IN THE DISTRIBUTED ENVIRONMENT

MKGLOB	Converts local system names into global names in the distributed environment.
RMGLOB	Releases global names.
MHNAME	Displays the name identifying the machine in a distributed configuration (see the chapter "Remote Service Access").
EXPORT	Exports all a system's global resources to the network's controller system.
NETNOSE	Examines the distributed environment: machine status, catalogue contents, etc.

PROCEDURAL LANGUAGE

A set of MCL statements is available for writing procedures. This set includes the structures typical of operating system command languages of this type.

LANGUAGE ELEMENTS

MCL accepts a number of keyboards and special symbols as well as a basic character set. This enables the user to write statements that include:

- variables (which do not require declaration, since they are always of string type)
- expressions (which may produce boolean or string results when evaluated)
- IF...THEN...ELSE, WHILE...DO, FOR...TO...DO, RETURN control structures
- BEGIN...END delimiter structures.

MCL enables the user to test particular conditions and run jobs automatically when these conditions are fulfilled.

BUILT-IN COMMANDS

The built-in commands are present within the Shell interpreter and are, therefore, available in all configurations. These commands are listed below, divided according to their functions. For further details, see the manual MCL, MOS Command Language User Guide.

WORKING DIRECTORY COMMANDS

SETWDIR	Changes the current working directory.
SHWDIR	Displays the path name of the working directory.

JOB CONTROL COMMANDS

BM	Executes a batch job.
LASTCOM	Sets the size of the history list.
TRACE	Allows a trace file for temporary information to be created.
PRIOR	Allows job execution priority to be modified.

COMMANDS FOR HANDLING VARIABLES AND STRINGS

SHVAR	Displays the contents of MCL variables.
LENGTH	Calculates the length of a string.
SUBSTR	Extracts a substring from a specified string.

COMMANDS FOR CONNECTING LOCAL NAMES TO GLOBAL NAMES

CONN	Connects a local (logical) name to a global (physical) name.
DISCON	Disconnects a local name previously connected to a global.
SHCON	Displays the current connections.

DISPLAY COMMANDS

ECHO	Displays the specified string.
WRITE	Displays the specified string on the system line of the screen.
SHNAME	Displays the name of the user currently logged in at the terminal.

EDITING A PROCEDURE

When writing an MCL procedure, the user must create a file with the system editor. The name assigned to the file is also the procedure name.

Each input line contains a series of statements, which must be separated from each other by a semicolon.

The MCL command EDPARSE is entered to check the syntax of an MCL procedure before executing it. EDPARSE highlights any syntactic errors within the procedure.

RUNNING A PROCEDURE

An MCL procedure is run by simply entering the name of the file that contains the procedure.

Parameter Passing

The user may associate positional and keyword parameters with the call (and thus with the file name containing the procedure), in order to specify options. This means of issuing parameters is also valid for MCL activation of programs written in Compiled BASIC or PASCAL+.

UTILITIES

A general introduction to the utility programs provided by MOS is given below.

SORT/MERGE UTILITY

The SORT/MERGE utility allows the user to:

- sort one or more input files according to specific keys, producing just one sorted output file (SORT)
- merge several input files, which have previously been sorted according to the same keys, into a single output file (MERGE).

Input files can be positional or keyed; the output file has the same structure of the input file.

User Interface

The user interface is made up of a language that is both interactive and procedural. This means that the user controls the SORT/MERGE program by using commands that contain the execution parameters. These may be grouped in command blocks or procedures.

The complexity of the command block varies according to the number and complexity of the required functions. The only mandatory elements in the command block are:

- the definition of the algorithm to be used (SORT or MERGE)
- the definition of the input characteristics
- the end-of-commands marker.

Functions

Apart from normal file sorting or merging, the user can request the functions listed below.

Record classification and selection: The user can classify input records according to types to be processed in different ways. On the basic of this classification, records of each type can be explicitly included or excluded from sorting; unless otherwise specified the default value (which is exclusion) is assumed; records that do not belong to any defined type are not included.

Moreover, it is possible to have sort keys in different positions within different types of records and still produce a uniform record layout in

output.

Sort key: Sort keys are fields chosen by the user in each record type to determine the record sequence for sorting.

The relationship between keys is hierarchical. There is one primary key on the basis of which the records are sorted. There can also be one or more secondary keys that determine further sorting levels within the first one.

Each key can be processed independently of the others in ascending or descending order.

Constants can be inserted as keys.

Selection of fields: For each input record type, it is possible to include or exclude fields explicitly from the input record format. The default value is always exclusion.

Totalling: In a record, totalisation fields can be defined so that, for groups of records with the same keys, only the first record in the group is output, and its totalisation field contains the sum of the values of the corresponding fields of all the other records in the group.

Collating sequence: Key fields of character-string type can be sorted using two character sets: ASCII and EBCDIC.

As an alternative, the user can provide his own collating sequence to meet special requirements.

Output record restructuring: The input record fields may be restructured in the output. This restructuring may involve the insertion of constants in order to obtain a different output format.

Repeated Runs

SORT/MERGE execution can be requested by the Shell SORTRUN command after the execution of the interactive phase, in which the command-block is entered. If a REPEAT command is entered in the command-block, it stores the whole command-block and assigns it a name. Thereafter, the user can just enter 'SORTRUN', followed by the name, to repeat execution. Thus, SORT/MERGE execution can be repeated a number of times with the same options.

Algorithms

SORT can be executed by means of two different algorithms:

- Minisort
- Maxisort

Minisort permits the sorting of files on hardware configurations with limited auxiliary storage.

Maxisort has higher-level features (in terms of execution speed) but needs more disk space and is typically used on hard disk configurations.

THE FCU UTILITY

The Floppy Conversion Utility (FCU) allows the user to exchange data between L1 MOS systems and other systems that use diskettes written to ECMA 58 or 67 standards.

The FCU utility is designed to enable data exchange with:

- IBM (or other) systems, which use 8" diskettes with a capacity of 1 Mbyte or 256 Kbytes, or 5 1/4" diskettes with a capacity of 1 Mbyte or 320 Kbytes.
- OLIVETTI M20 systems, which use 5 1/4" diskettes with a capacity of 1 Mbyte or 320 Kbytes. (The M20 system does not write to ECMA standards, but a special conversion utility is available.)
- M30/M40 BCOS II systems.

User Interface

FCU is a normal Shell command available to all users. When activated, it provides two interactive functions:

- READFLOP, which reads data from a diskette written either to ECMA or IBM standards.
- WRITEFLOP, which writes L1 data files onto a diskette in either ECMA or IBM standards.

Characteristics

File types

FCU allows the exchange of data files only, which must be recorded using ASCII or EBCDIC codes, or a conversion table defined by the user by means of the auxiliary FCUTAB utility.

Only a single file may be written on a diskette when using FCU, but all

the files on a diskette can be read by repeating the FCU command. The file to be converted can reside on several diskettes.

Identification

When data are written on to a diskette using the WRITEFLOP function, descriptive information on the data and their structure is also included.

The READFLOP function expects to find similar data written on the diskette to be read.

Trace file

If required, a trace file may be selected to contain a record of all conversion operations performed during the current activation of FCU. This includes all the identification data written on the diskette itself such as:

- the function performed (READFLOP or WRITEFLOP)
- the record length of the data files
- the system with which data are exchanged
- the data format used
- an indication of the result of the conversion (correct or abnormal termination).

THE TCU UTILITY

The Tape Conversion Utility (TCU) allows the user to exchange data between L1 MOS systems and other systems that use magnetic tape written to ECMA 13, OLIVETTI 32, IBM or UNLABEL standards. (Only the UNLABEL is currently supported.)

This utility supports magnetic tape written with density 1600 bpi.

User Interface

TCU is a normal Shell command available to all users. When activated, it provides two interactive functions:

- READTAPE, which reads data files on magnetic tape.
- WRITETAPE, which writes L1 MOS data files onto magnetic tape. The writing operation may either start at the beginning of the tape, thus overwriting any existing information, or follow on from data already present.

Characteristics

File types

TCU allows the exchange of byte-stream data files only. Both data files and object files are allowed, as long as they are written using ASCII or EBCDIC codes, or a conversion table defined by the user by means of the auxiliary utility FCUTAB (see the section "The FCU Utility" above). These files can be multitape.

Trace file

If required, a trace file may be selected to contain a record of all conversion operations performed during the current activation of TCU. This includes all the identification data written on the tape itself such as:

- the function performed (READTAPE or WRITETAPE)
- the name of the file
- the data format used
- an indication of the result of the conversion (correct or abnormal termination).

4. HANDLING FILES AND VOLUMES

MOS handles input/output on peripherals that are accessed directly (disks or diskettes) or sequentially (printers, terminals, SCT, MTU) by means of File System Management, the MOS component that allows the user to handle input/output in 'device-independent' mode.

This chapter illustrates the basic concepts, the characteristics and the operations regarding "objects" of the file system.

Each machine in a cluster configuration, consisting of several interconnected homogeneous systems, has its own File System Management. This is described in this Chapter.

The set of files on a machine is known as the "local file system".

All the "local file systems" together make up the "global file system" (see Chapter 7, "Networks").

BASIC CONCEPTS

The following outlines the information necessary for operating and applying the system.

NAMING THE FILE SYSTEM OBJECTS

All objects of the file system are referred to by a unique name that is associated with them when they are created. The objects that may be named are:

- devices
- files
- directories
- volumes.

A symbolic name is a character string that may contain any of the following characters:

A ... Z

a ... z

0 ... 9

The 'national' characters typical of some keyboards, and the following

special characters, are also permitted:

. _ $

The user may select the symbolic name for an object that he is allowed to create.

DEVICES

A device is physically connected to the system. All devices have predefined names. These are:

- FLn : 8" diskette unit (n = 1, ..., 4)
- MFn : 1 Mbyte 5 1/4" diskette unit (n = 1, ..., 4)
- MFn : 320 Kbyte 5 1/4" diskette unit (n = 5, ..., 8)
- HDn : hard disk (n = 1, ..., 8)
- SC : streaming tape cartridge
- MTn : Magnetic tape (n = 1 or 2)
- TTYA, TTYB, ..., TTYZ, TTY0, TTY1, ..., TTY5, RT01, RT02, ..., RT32: work stations A, B, ... (following the English alphabet), 0, 1, ... (see Chapter 5, "Work Station Management".)
- SYSPRTn : system printer (n = 1, ..., 8)
- VD : volume in memory (see below, "Memory Volume" Section.)

The user is allowed to define synonyms for pre-defined device names.

FILES

A file is a collection of data stored as a logical sequence of bytes. The data may be logically grouped in records. The data type may be:

- a program, in source or object format
- text
- application data

The user may access a file in the following ways, depending on its types of organisation (see the section "File Attributes and Basic Operations - Organisation and Access Methods"):

- sequentially
- directly in byte-stream mode

- directly in keyed mode through its key or order number

A file may be held temporarily or permanently on any type of physical support, such as:

- disk (hard disk or diskette)
- tape (SCT or MTU)
- terminal
- printer.

The user is freed from the problems of handling the physical support and can concentrate on the data contained in the file.

File System Management enables identification of a file by a user-selected name, which is communicated to File System Management at file-creation time.

DEVICE INDEPENDENCE

This characteristic allows a file to be stored on any physical devices; it may then be accessed independently of the physical characteristics of that device. Naturally, this depends on the file's organisation: for example, a magnetic tape is a sequential device that can be used to store files of any type.

When using this characteristic, the input or output of a program may be redirected to any device that will accept the file organisation concerned. To do this, the user has only to connect (CONN command) the logical file name to the predefined device name or physical file name that is to be the storage medium.

DIRECTORIES

The set of files on disk is hierarchically organised in a tree structure consisting of a set of nodes and branches. The nodes contain information and the branches define the relationships between nodes of different levels. The end nodes, known as leaves, are the files, and the intermediate nodes are the directories. Directories allow the logical grouping of files.

A directory contains the names and the information for locating all the objects that depend on it, and may also contain other directories known as subdirectories.

Figure 4-1 gives an example of the hierarchical storage structure.

Note: Directories and volumes are shown in the figure. The volume concept is described later.

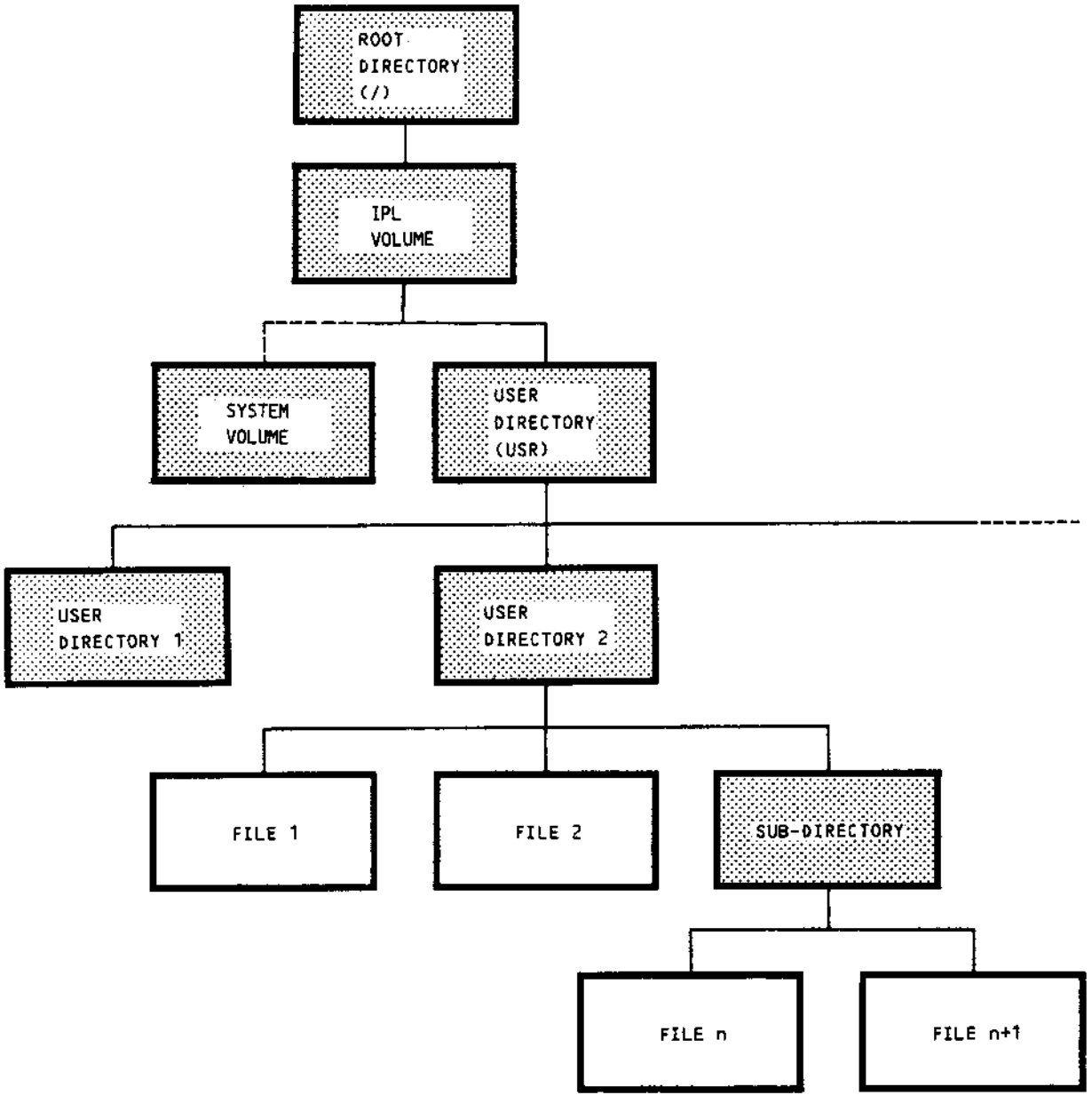


Fig. 4-1 Hierarchical Storage Structure

'/' Root

The root of the tree, known as '/', is loaded into memory when the system is initialised and contains the names and the information necessary to access its subdirectories.

System Volume

The system volume contains programs and data used by the system. These may not be accessed by the user.

User Directory (USR)

The user's directory (USR) contains the names of all the system users and the information required to access them. This directory is only present on multi-user systems. It can be created at installation time using the MKLOGIN command, which creates all the login support structures.

Device Directory (DEV)

The device directory (DEV) contains the names of all the available devices. These are indicated with the names described previously in the Section "Devices". This directory is created automatically at system initialisation.

Home Directory

This is the directory in use immediately after the user has logged on to the system.

Working Directory

The working directory is the directory under which the user is working. Initially, this is the user's home directory, but it may be changed by using an appropriate command.

Parent Directory

The parent directory is that which is directly above the working directory in the hierarchy.

Operations Allowed on a Directory

- create a directory
- remove a directory
- change the name of a directory

- copy one directory under another
- delete the contents of a directory
- list the contents of a directory.

PATH NAMES

The hierarchical structure illustrated above enables each user to group his files in his own directories, giving each one a path name that is unique in the system.

The path name represents the sequence of nodes that the system must pass through, starting from the root, in order to reach a particular file.

It consists of a sequence of names separated by the '/' character. Each name may be the name of a volume (see below) or a directory. The last name is that of the object that the user wishes to access. The file name is always in the last position. Figure 4-2 illustrates how the system starts from the root, identified by '/', and follows through the subsequent nodes to reach the file with path name:

/IPL/USR/CHARLIE/PROG.1

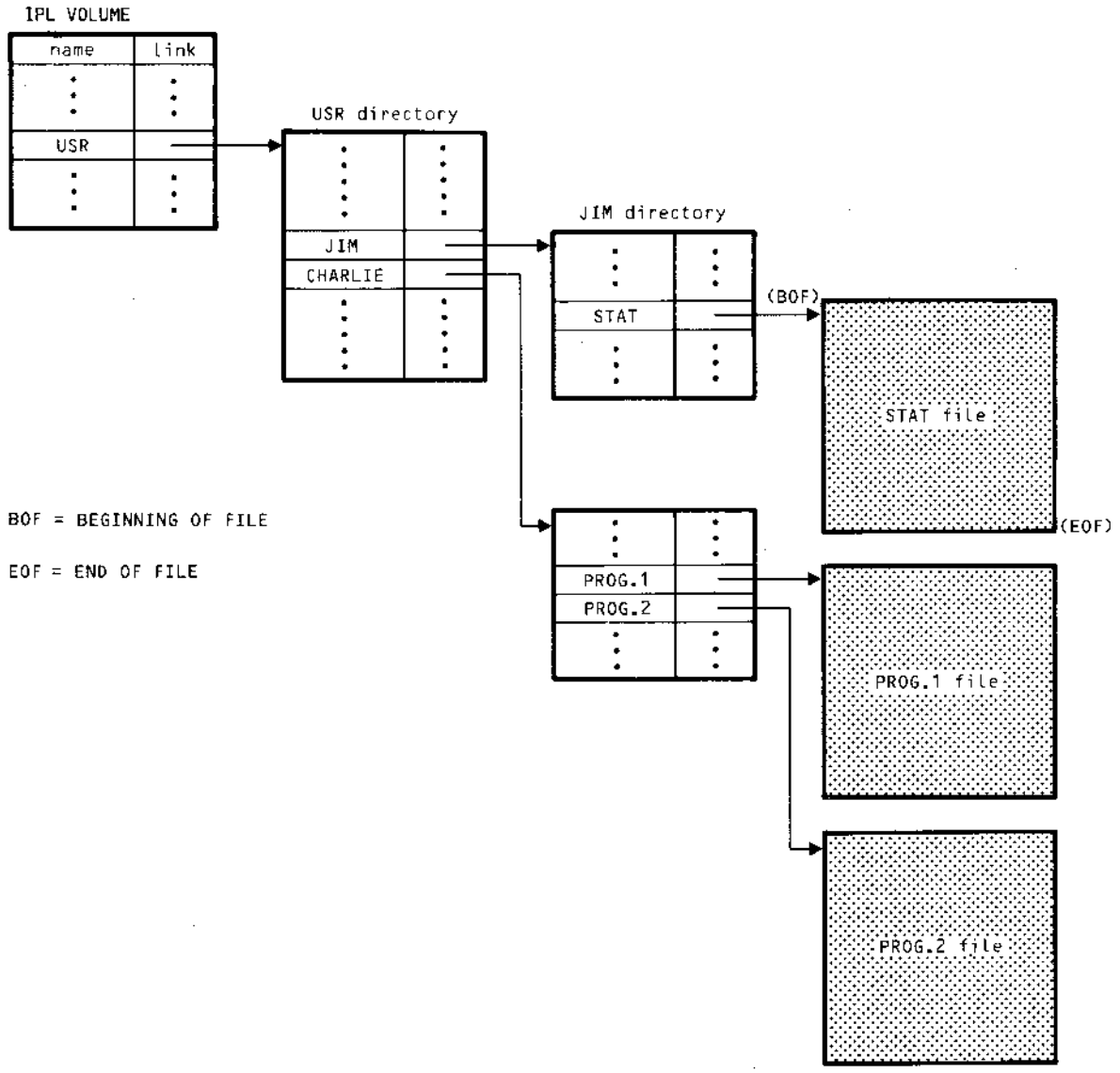


Fig. 4-2 File Search

The user may provide the entire path name or a partial path name to identify a file.

Entire Path Name

This is characterised by the initial '/'. In this case, the system starts scanning the tree from the root.

Example: /IPL/USR/CHARLIE/PROG.1

Partial Path Name

This is characterised by the absence of the initial '/'. In this case the system assumes that the file or directory is to be looked up under the working directory.

Example: PROG.1, with working directory "CHARLIE".

Special Characters

The following graphic symbols or special characters have a specific meaning in a path name.

. This character identifies the working directory. It allows the working directory to be specified as a parameter within a user procedure; for example, if 'CHARLIE' is the working directory, then the path name ./PROG.1 is extended to /IPL/USR/CHARLIE/PROG.1 at run time.

.. This character identifies the parent directory; for example, if CHARLIE is the working directory, the file STAT under the directory JIM is identifies by the path name ../JIM/STAT.

VOLUMES

A volume is a collection of file system objects stored in logically contiguous disk areas of a size defined by the user during the creation phase. This size cannot be modified afterwards. These objects may be other volumes, directories or files. A volume can extend over several disks, which must be made logically contiguous with interventions on the configuration file or using the Shell command LINKDEV.

Subvolumes

A volume may be divided into subvolumes via the MKVOL command, as shown in the following figure.

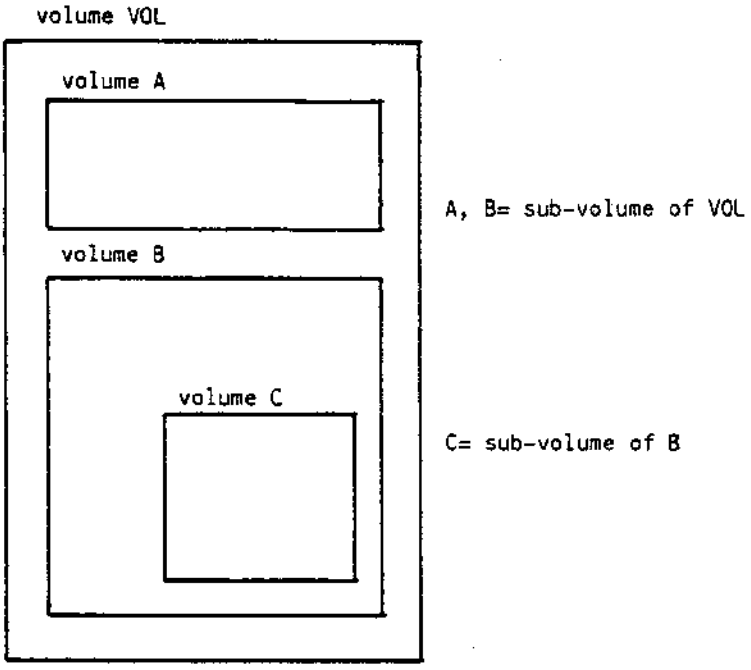


Fig. 4-3 Volume Structure

Each volume contains:

- a volume descriptor that provides information regarding the size of volume, the maximum number of files in it and its name
- all file descriptors belonging to the volume. These descriptors contain the file attributes (see the section "File Attributes and Basic Operations")
- the root directory (.)
- all files and directories belonging to the volume.

No information is given about physical addresses outside a volume. In this way a file may be moved within a volume, and a volume may be physically copied onto another part of the disk or onto a different medium (tape) without having to give a physical address.

Both a volume and a directory represent a grouping of other files and directories. No distinction between a volume name and a directory name is made in a path name.

The difference, as regards the physical organisation of the disk, is recognised automatically by File System Management.

Operations Allowed on a Volume

The user may create a volume on a fixed or removable device (see below), change the name of a volume, delete a volume from any device and copy volumes to or from any magnetic device (HD, diskette, tape), or optimise the use of the magnetic supports by compacting the space occupied by the volume.

REMOVABLE VOLUMES

A removable volume is a storage medium, such as a diskette, that may be removed from its drive. If removed, the data contained in it are no longer available to the system.

All directories, volumes and files on a removable volume are referred to using a standard path name; thus, a removable volume that has been mounted is treated as a fixed volume when carrying out operations on its contents.

A removable volume can be logically connected or disconnected (MNT, UNMNT) at any point in a directory tree (see Fig. 4-2). The path name specified in the command indicates the directory (or volume) beneath which the mount or unmount operation is to take place; for example, to mount volume VOL1 beneath the directory SUB-DIR the path name to be specified is:

/IPL/USR/CHARLIE/SUB-DIR/VOL1

The last name specified (that is, VOL1 in our example) is the name used to connect the volume, through which it is possible to access the directories and files contained in VOL1. This name may differ from the physical name of the volume found in its descriptor. This feature allows access to volumes whose name is not known.

In the example just given, the file

/DIR1/FILE1

contained in VOL1 (after this has been connected) is accessed through the path name:

/IPL/USR/CHARLIE/SUB-DIR/VOL1/DIR1/FILE1

If SUB-DIR is the working directory, FILE1 is accessed through the partial path name:

VOL1/DIR1/FILE1

MEMORY VOLUME

When the MOS operating system is loaded, a volume is created in memory, called "/". This means that a contiguous area of memory is reserved for file cataloging. The data recorded in this volume are on a virtual peripheral with direct access.

The number of files that can be contained in the memory volume is predefined at configuration time.

If, for example, the presence of files in memory is not an important feature in a particular configuration, a volume can be created in memory containing only those system files that must reside in memory.

As the memory volume is resident on a volatile support, it is used to store files that do not need to be kept for long periods of time.

For example, with regard to system files, these are the files describing the peripherals and those containing information on program execution. With regard to user files, these are the temporary files whose access time is considerably faster than those stored on disk.

Any type of file that can be handled on disk (byte-stream, positional, keyed, directory) can be created in the memory volume.

SYSTEM DISK REMOVAL

The memory volume created when MOS is loaded is the only volume in the system that cannot be removed.

When it is created, the system volume is logically connected to its main directory with the name /IPL.

In the same way, any other volume (resident on disk) can be logically connected to the memory volume by the user.

All the volumes resident on removable supports (including system volumes) can be removed and replaced without limit.

FILE ATTRIBUTES AND BASIC OPERATIONS

The most important attributes associated with a file on disk are its:

- physical name
- allocation unit
- type of organisation.

These attributes are defined by the user at file-creation time and they my be modified.

Other notable file attributes are:

- a user-defined file type, having a special meaning for the user
- the date of file-creation
- the date on which the file was last updated and the date on which it was last accessed.

ALLOCATION UNIT

The allocation unit is the contiguous space on disk (expressed in number of bytes) that the user wants to allocate to a file:

- at file creation
- when expanding the file.

The importance of the allocation unit lies in the fact that its value may influence the time necessary to access the disk. For greater detail on the correct use of this parameter, see the manual MOS Programmer Guide.

ORGANISATION AND ACCESS METHODS

A file may be organised in one of the following modes:

- Byte-stream
- Positional
- Keyed

The records of positional or keyed files are of fixed length.

Each mode of organisation is described briefly, together with operations that may be performed on the various types of files.

Byte-stream File

A byte stream file is a sequence of bytes, identified by their position. All bytes within the file extent are valid.

The user may read or write a sequence of bytes by specifying the number of bytes in the sequence and the position of the first one. When writing, it is possible to append bytes to the end of the file or to write a sequence of bytes at any point in the file, possibly overwriting a part of it.

Positional File

A positional file is a set of records of fixed length with which a numeric identifier, greater than or equal to zero, is associated when the record is written.

The concept of record validity exists in this type of file: the only records that are valid and which thus may be accessed within the file extent are those that have been written since file creation and have not been deleted by the user.

It is possible to:

- write a record, or a sequence of records, from the beginning, from the end (append), or from any position within the file
- read a record, or a sequence of records, from a given position within the file (in ascending or descending order)
- update a record
- delete a record or a sequence of records.

When a record is deleted, it may no longer be accessed even though it is still physically present in the file.

Keyed File

A keyed file is a set of records of fixed length, each one identified by a user-defined key. It may contain two types of key:

- A primary key, which is compulsory for record creation. This may be a field internal or external to the record
- One or more secondary keys, which are optional; each one of these may be any field within the record.

A primary key refers to one record only, whereas secondary keys may be duplicated. Since it is possible to access records through various fields, a file may be seen as ordered in different ways, depending on the key used.

Keys are held in data structures known as indices; these allow access to the file (for further details see the manual MOS Programmer Guide). Each type of key has:

- a primary index generated at file-creation time
- zero or more secondary indexes, which may later be created or deleted by the user.

Records in a keyed file are identified by specifying either the primary or a secondary key.

All operations allowed for positional files are possible together with the following:

- Reading a record via a primary or secondary key.
- Reading record sequences in ascending or descending key value, starting from any key within the file.

- Reading a record by providing the prefix of a primary or secondary key, with the possibility of reading the preceding or subsequent record.

- Updating a record via the primary key.

- Substituting one record for another with an identical key.

- Locating the first of a group of records having duplicate secondary keys; through this it is possible to read by means of the key itself the records in the group in sequence.

- Inserting new records. When a new record is inserted, a new key is created within the primary index; this determines the point in the file at which the record is inserted.

- Deleting a record by specifying the primary key: when the record is deleted it remains physically present in the file but cannot be accessed.

Sequential Access

All files present in the file system can be sequentially accessed.

The operations allowed on the file are writing records to be appended at the end of the file, and reading the record immediately following the last one read.

OTHER FILE OPERATIONS

In addition to access operations, File System Management allows other operations on files. For the purpose of these operations files are considered as global entities and not as records and bytes.

The first and most important operation is file creation. This entails the allocation of space on the storage medium on which the file must reside. The space thus allocated is identified by the user-supplied file name.

When creating a file, the user must define the file attributes as well as supply the directory under which the file is stored.

Having created a file, it is possible to open it. When a file is no longer required during a job, it should be closed.

In addition to the above the user may:

- compact a file

- identify the type of file organisation

- change the type of file organisation

- rename a file
- copy a file
- reduce the file length
- empty a file
- remove a file.

FILE-TYPE CONVERSION

A file may be converted from one type of organisation to another. The following conversions are allowed:

- From a keyed file into a positional or byte-stream file
- From a positional file into a keyed or byte-stream file
- From a byte-stream file into a positional or keyed file.

Note

A file having a certain type of organisation may be accessed in the same mode used for a lower level organisation. For example a keyed file may be accessed in byte-stream or sequential mode.

When operating in this manner, the auxiliary structures present in the file organisation (bit map, indices, etc.) are not updated. The data set may become inconsistent with these structures; this may cause unexpected changes in their behaviour pattern when the file is next accessed.

'ALIAS' FILES

An 'alias' file contains the name of another file. Each time an I/O operation is requested on the 'alias' file the system executes it on the file with the name contained in the 'alias' file.

For example, let us suppose we have the following two files:

/A/B (containing any data)

/X/Y (containing the string /A/B)

The /X/Y file is an alias of the /A/B file.

A read/write request on the /X/Y file has the effect of making the same request on the /A/B file, and the latter is always the object of the operation.

If, however, the file /A/B/C exists, meaning that the /X/Y alias file refers to a directory and not a file, it can be referred to be indicating the complete path name (/A/B/C) or, using the alias, the path name /X/Y/C.

FILE SHARING

The File System Management allows several users to access files concurrently. This feature is important as a shared file can be used for communication between users.

The system provides a "lock" mechanism that regulates concurrent access to a file thus avoiding errors caused by interference.

One of two types of lock may be used:

- File lock, which excludes the possibility of file sharing.
- Record lock, which allows file sharing.

File Lock

The file lock is of "exclusive" type. When a user opens a file specifying this type of lock, it may not be opened by another user. The user who has imposed this lock becomes the temporary "owner" of the file. The user may request a lock when opening a file only if the file is not being used by other users at that moment.

A file lock may be applied to files having any type of organisation.

Record Lock

A record lock is of "modify" type. This type of lock is requested because the user wishes to modify the record. When imposed, the record may be accessed by other users for read operations only.

If a user modifies a record without previously locking it, other users may access the record at the same time and the data will be confused.

The record lock (and file sharing) allows data to be exchanged and shared between various users.

This type of lock may be imposed only on files whose records may be accessed in direct mode, that is, on positional or keyed files.

Time Out

A time out may be specified when requesting the opening of a file or access to a record (with or without lock). This value expresses the maximum time that the user making the request is willing to wait in the event that the file cannot be immediately opened or the record accessed. If the cause of the delay is not removed, the request is automatically cancelled when 'time out' elapses.

This delay can be caused because the user is trying:

- to open and lock a file that has been opened by other users
- to open a file that has been locked
- to lock a record that has already been locked by another user.

Lock Release

All locks present in a file are released when the file is closed. A lock on a record may however be removed whenever the user wishes to do so (that is, when it is no longer necessary).

Phantom Lock

Users working on keyed files can reserve one or more keys before starting their activities.

USING THE FILE SYSTEM BUFFERS

The write operations requested by File System Management are executed in synchronous and asynchronous mode.

SYNCHRONOUS MODE

Write operations in synchronous mode are executed as soon as they are requested. When they have been completed control is returned to the requesting program.

ASYNCHRONOUS MODE

If a program requests an asynchronous write operation, it continues execution while the request is being carried out by the File System.

These two modes of writing use system buffers, the number and size of which are defined at configuration time. An alternative to these are the private buffers, also defined at configuration time. In this case a private buffer is associated with the file to be written (MCL SETBUFF command), thus avoiding competition from other users and accelerating the operation.

FILE SYSTEM MANAGEMENT INTERFACE

When carrying out I/O operations in application programming, the user accesses File System Management indirectly through programming language statements, which cover many of the appropriate features and mechanisms described earlier.

The run-time support of each programming language interprets I/O requests from application programs and expands them into sequences of File System Management interface primitives. These interface primitives may be accessed directly only by system programs written in PASCAL+.

5. WORK STATION MANAGEMENT

This chapter describes the operating and programming characteristics of MOS components relating to Work Stations. By "Work Station" (WS) is meant the group of peripherals that enables the operator to communicate with the system.

The components described in this chapter are Work Station Management (WSM), the RS232/CL and encryption drivers, the software for handling the CAT work station, and the software for handling Personal Computers (PCs) as L1 work stations.

WSM provides basic facilities for work stations consisting of a screen/keyboard combination and optional extras such as printers, badge reader/writers, etc.

The RS232/CL driver enables any peripheral equipped with an RS232 serial interface to be linked to MOS systems, thus extending the range of WS peripherals that can be linked to MOS. The encryption driver provides features for ensuring data security.

The software for CAT handling provides a special interface for this type of WS, used typically in banking environments.

WORK STATION MANAGEMENT

Work Station Management handles, and serves as interface for, a vast range of WS connected to the system in different ways.

- Integrated local work stations (KDC), (connected directly or using ELB 1381/1382), based on alphanumeric or graphics terminals, monochromatic or colour.

- Local or (via modem) remote work stations (ELB 3683/3684), connected to the system through the MUX controller (known as WSL1), based on alphanumeric monochromatic terminals.

- OLIVETTI Personal Computers (PC) (M19, M24, M24 SP, M28), used as MOS work stations, permanent or dynamic, local or remote. Note that PCs can be used in this way through the use of a MOS terminal emulation program running on the PC.

- M31 Systems (WSG 3622) used as permanent MOS work stations; local or remote, based on graphics or monochromatic terminals.

- M30 Systems (WSG 3623) used as permanent MOS work stations; local or remote, based on graphics and colour terminals.

- VT 100-like work stations (for example: WS 584), provided with terminals functionally similar to those of Digital Equipment (alphanumeric and mono).

The last three types of WS may be linked to the system by means of the SIC or MUX interface. The link may be direct (using RS232 PERIPH or CURRENT LOOP cable) or through a modem.

Most of the above work stations may be physically linked to the following optional modules: printer, PIN pad, badge reader/writer, cheque reader/writer and cash adapter. The WSM handles (in cooperation with the emulation programs, in the case of the PC) the screen/keyboard unit of all the WS listed above, as well as:

- all the other WS units, in the case of KDC and WSL1

- the WS printer; with certain types of emulators on the PC

- the sole function of hard copy to the printer, in the case of M30/M31 systems.

The WSM interface is placed between the user and the hardware to provide:

- logical programming interfaces that hide from the user physical characteristics, peripheral connection modes, and their associated initialisation/shutdown routines.

- programming interfaces, mostly "device independent", for each of the WS peripheral classes; interfaces that, while being logical, also allow full use of the specific features of the peripheral.

- more sophisticated operating interfaces than those of the peripheral (for example, in the case of terminals, automatic display (echo) of information input via the keyboard).

In view of the multiplicity of WS peripherals, the WSM is made up of as many sub-components, called drivers, as there are types of WS peripheral: the terminal driver, the printer driver, etc.

There is an instance of each driver for each of the WS peripherals handled by the WSM; the functions of the driver - and therefore the peripheral - are available uniquely to the user environment(s) provided in the activity selection menu associated with the terminal and applications activated by it.

For further information on the WSM programming and operating interface refer to the manuals MOS Programmer Guide, MOS Introduction to System Programming and MOS Operating Guide.

For further information on PCs connected to MOS, refer to the section "Software for Handling the PC as a Work Station" later in this Chapter.

TERMINAL DRIVER

The terminal driver allows users to send data/commands to all the work stations listed above.

Types of Screen/Keyboard

The driver handles the WS keyboards transparently, in particular the following OLIVETTI work station keyboards: the management and DP environment keyboard, the scientific/technical environment keyboard, and the multifunction keyboard. The latter combines the features of both the other two keyboards.

Similarly, the driver transparently handles different types of screens, with different physical dimensions (15", 14", 9" trivalent and 5") and different display modes (alphanumeric or graphical, monochromatic or colour). Thus, an application remains unchanged for screen/keyboard units having the same features.

General Characteristics of the User Interface

The driver allows use of the same screen/keyboard units by a number of programs of the same application, and therefore belonging to a single user, the operator of the work station that activated them.

The work station screen may, in fact, be divided into two subsets of contiguous lines called windows. At any moment only one window is active, that is, able to receive data/commands: this is the one to which the keyboard is logically connected, and which contains the cursor.

The driver handles the despatch of commands and data introduced by the operator (using the function and alphanumeric keys) to the program

associated with the active window, and despatches commands and data from the program to the screen (Fig 5-1).

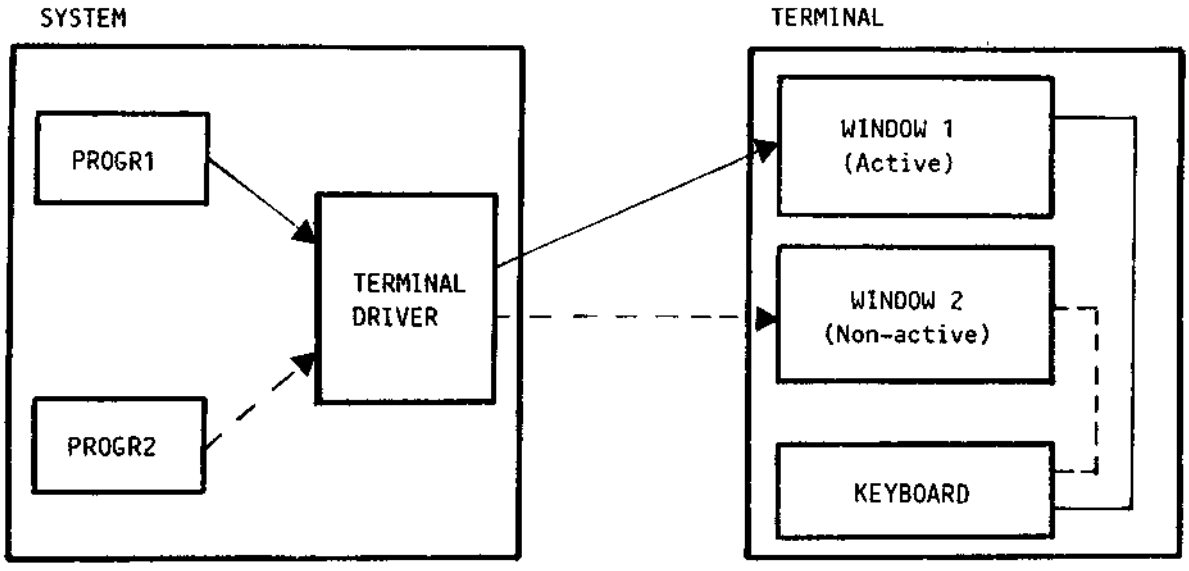


Fig. 5-1 Despatching Data from Program 1 to Window 1

Using the keys causes ISO characters to be sent to the program (normally two, in the case of function keys); the program may send both data (ISO characters and/or graphics) and commands.

In the case of VT-like, M30 and M31 WS, only one window is available; in the case of the PC, the availability of two windows is dependent on the particular emulator in use.

As regards the keyboard, the driver offers a number of basic functions (hard copy, end-of-string recognition, etc.) associated with the use of certain function keys, and includes the possibility of reconfiguring the function keys in order to customise the keyboard. More precisely, independent of the keyboard type, the driver offers:

- for the function part:
  - the possibility of assigning application-type functions to certain keys (F1 - F28)
  - the possibility of redefining, in the configuration phase, the relationship between function keys, basic functions, and application functions

- the possibility, in some cases, of redefining the relationship between function keys and basic functions while the application is running.

- for the alphanumeric part:

  the handling of many national keyboards by means of translation tables configurable for each work station.

Programming interface

The driver interface is visible only to the PASCAL+ language. Other application languages translate application I/O requests into a series of terminal driver interface primitives.

The programming interfaces supplied by the driver are characterised by their flexibility in allowing, depending on the needs of the application, both simple and sophisticated use of the peripherals. The user can either regard the screen as a sequential device (made continuous by the driver default features: scrolling, end-of-line handling, etc.) or use the complete range of commands provided for the peripheral (cursor control commands, etc).

In either case, transfer of data between program and work station (and vice-versa) takes place in TTY (teletype) mode: one string at a time.

Handling an entire screen page, and the consequent transfer of data at page level (instead of string level), is the task of VISA.

The driver interfaces are used by the File System in the case of I/O operations on the terminals.

The interface provided is not entirely device independent, owing to the variety of terminal connections (local or remote) and to their different characteristics (OLIVETTI terminals, VT-100-like terminals, OLIVETTI systems or PCs used as terminals).

To be able to use the programming interfaces provided by the driver it is necessary to know the system identifier of the window or windows available to the application.

This is found in the program "context"; that is, in an information structure that describes the program execution environment (the working directory, the standard input and output units, etc). The standard input and output units are by default the keyboard and the terminal screen available to the program. The identifier of the first (or only) window coincides with that of the WS.

Output features: Both data and commands can be sent to the terminal. If the user uses a VT-100-like terminal he may use only the set of commands supplied by the VT 100 manufacturer.

If he uses a work station built by OLIVETTI (KDC, WSL1), or an OLIVETTI system used as WS, he has available two sets of commands: OLIVETTI and VT-100-like.

The OLIVETTI set of commands is largely the same for all the WS and OLIVETTI systems used as WS. They may be grouped into:

- cursor control commands
- commands for displaying characters on screen in alphanumeric and in various graphical modes
- visual attribute handling commands at character level
- delete commands for part of the window
- various commands (activation of an acoustic signal, lighting up or switching off keyboard lights, graphical commands, etc.)

The VT-100-like command set permits operations that are less sophisticated than the OLIVETTI set, but are functionally compatible with those of VT 100 terminals. Availability of this subset simplifies the conversion of application and system software from VT 100 terminal software to L1 line software.

Input features: It is possible to operate in one of two states depending on the state of the window: normal or "raw". The "raw" state is utilised only by specialised applications and is therefore not described in this manual.

In the normal state, the terminal may be asked for one or more characters. If asked for a string of characters, the driver echoes each character, and does not pass on to the program the characters corresponding to the function keys recognised by the WSM. If asked for only one character, the driver does not echo the character, and passes on to the program the characters corresponding to the function keys.

Operating interface

Immediately after switching on the system there is only one window on the screen. This window, called the "father", has the same dimensions as the screen. When a second is created, called the "daughter", a horizontal line appears (which does not coincide with a screen line) separating the two windows.

When the "daughter" window is closed, the separating line disappears and all the lines that formed it are occupied by the contents of the "father". The "daughter" window is normally used by the system software (for example by Shell) for handling asynchronous events, and consists of only one line. If a window has already been used, the cursor returns to the last position it occupied before it left that window.

For introducing commands and data the operator has function and alphanumeric keys, the latter in the appropriate national version. Some function keys are interpreted by the WSM, while the interpretation of others (for example, some cursor movement keys) is left to the program.

In this way the operator can use an operating environment jointly set up by the WSM and the application.

The operator can also introduce data in the absence of the screen cursor. The characters keyed-in are stored in the buffer, and echoed only when the application issues a read request. This prevents any characters sent by the application appearing on the screen and interfering with those introduced by the operator (and displayed by the driver).

Characters are echoed in the active window, starting from the point on the screen at which the cursor is positioned, and using the keyboard alphabet of the work station.

The ability of the program to turn echo on or off allows the operator, when echo is on, to make a visual check of the data displayed; when echo is off, he can maintain the secrecy of reserved data (for example, a password).

The basic features of the WSM accessible to the operator by means of the function keys are as follows:

- Changing the active window by pressing a configurable key (**CH.WIND**).

- Terminating a string by pressing the **CR** key (in the case of strings longer than one character); any attempt by the operator to use a different terminator is rejected by the WSM, which emits an acoustic signal.

- Controlling the scrolling of data across a window by using two non-configurable function keys: the first (**CNTRL S**) stops the scrolling movement activated by the program, while the second (**CNTRL Q**) restores it.

- Requesting hard copy by pressing a configurable key (**H.COPY**).

- Interrupting the program by pressing a configurable key (**HALT PGM** , in the case of a multifunction keyboard). In the Shell environment, for example, this allows the current application to be suppressed or suspended.

- Signalling the end of data entry by pressing a non-configurable key (**CNTRL D**).

The WSM also allocates an identifier to each terminal. Terminals are divided into two classes, depending on whether they belong to a WS connected to a system permanently or dynamically. Terminals of the first class are given an identifier of the form TTYx, where x is in the range A - Z or 1 - 9; those of the second class are given an identifier of the form RTnn, where nn is in the range 00 - 32.

THE PRINTER DRIVER

The printer driver handles a vast range of work station printers with different characteristics and functions: general purpose, high quality, and specialised printers (for example, those designed for banking environments).

It can therefore handle each of the sub-devices that may be components of the printers indicated above: continuous feed, left-hand, right-hand, discrete paper module feeders, passbooks with magnetic stripes, etc.

Programming Interface

The driver programming interfaces are visible only to PASCAL+ users. To use them it is necessary to know the system identifier of the WS to which the printer is connected. For this information refer to the "Terminal Driver" section, above.

Interpreted BASIC, Compiled BASIC, and COBOL ICE translate application print requests into the appropriate series of driver interface primitives.

The driver interfaces permit:

- writing on continuous or discrete paper modules
- reading and writing magnetic characters, if the printer is equipped to do so.

Unlike the terminal driver, the printer driver allows sharing of peripherals between different work station users. Use of the printer is allowed to the programmer who first starts a work session. If other users require the printer they must wait until closure of the current session.

Like those provided by the terminal driver, the printer driver interfaces permit the sending of data or commands, and are characterised by their flexibility. In fact, the program may either use the commands supplied by the driver ("transparent" use of the peripheral) or send data directly to the default sub-device (the printer sub-device chosen at firmware level), adding the appropriate line spacing commands (Line Feed).

In the case of transparent use, the codes that can be sent depend on the printer type. They include, for example, commands for sub-device selection, and for setting document format, line spacing, and character attributes.

Operating Interface

The driver handles operator booking LEDs (on printers shared between a number of operators). For further details refer to the manuals for each printer type.

BADGE READER/WRITER DRIVER

The badge reader/writer driver handles readers and reader/writers of plastic cards (badges) having two magnetised tracks, and provides the user with device-independent logical I/O interfaces.

The read-only unit reads information exclusively from credit cards, while the read/write unit reads/writes information on credit cards, magnetic forms, or passbooks having compatible measurements.

The programming interfaces provided by the driver completely hide the peripheral commands; the user is therefore only involved in sending requests for reading/writing data on two magnetic tracks (track 2 and track 3 - track 1 is for reading only). Peripheral sharing is not possible.

The driver programming interface is visible only to users of PASCAL+. To use it requires knowledge of the system identifier of the WS to which the peripheral is connected. For more details refer to the "Terminal Driver" section.

COBOL allows access to VISA badge reader/writer interfaces.

The application program must instruct the operator on the mechanism of inserting the badge into the peripheral.

CHEQUE READER/WRITER DRIVER

The cheque reader/writer driver handles the peripheral units for reading or reading/writing MICR documents, and offers the user specific logical I/O interfaces.

The programming interfaces provided by the driver allow the transmission/reception of data and the sending of commands. The latter are, however, sent in parametric mode, thus preventing the user having a direct knowledge of the peripheral commands.

It is possible to request in this way:

- Insertion of the document
- A fixed and/or variable stamp (the latter not with magnetic characters). By "stamp" is meant the MICR characters at the foot of the document, showing the customers bank code, etc).
- Manual ejection of the document (only for the cheque writer) or document ejection with choice of ejection tray (sort).

Note that coding of data from ASCII to MICR or vice-versa is done under the control of the peripheral, not the driver.

The driver programming interface is visible only to users of PASCAL+. To use it requires knowledge of the system identifier of the WS to which the peripheral is connected. For more details refer to the "Terminal Driver" section. COBOL allows access to VISA badge reader/writer interfaces.

PIN PAD DRIVER

The PIN Pad driver handles the PIN Pad (Personal Identification Number) peripheral and offers the user specific logical I/O interfaces.

The PIN Pad is made up of a numeric keyboard, two buttons and two LEDs. It enables the users of the system to which it is connected to key in and transmit a code, or secret number, which is verified by an application program before proceeding with operations on the system itself. On system start-up, the peripheral runs autodiagnostics, checking ROM and RAM.

The PIN Pad driver indicates activation of the unit by lighting up the green LED. At this point, the user may key in his identifying number. Having done this, he must press the green button to confirm, or press the red button in order to avoid transmission of an error message (indicated by the red light coming on), thus allowing repetition of the keying-in operation. The input is handled by the driver buffer.

The interface primitives allow both the reception of data and the sending of commands, the latter transparently. Among these are control commands, as follows:

- Switching the green and red LEDs on and off.
- Enabling the keyboard.
- Requesting the result of the autodiagnostics.

The driver programming interface is visible only to users of PASCAL+. To use it requires knowledge of the system identifier of the WS to which the peripheral is connected. For more details refer to the "Terminal Driver" section.

COBOL allows access to VISA badge reader/writer interfaces.

CASH ADAPTER DRIVER

The cash adaptor driver handles the cash adaptor, and offers the programmer specific I/O interfaces.

The cash adaptor is a compact system of hoppers for loading and counting banknotes that enables them to be dispensed at high speed, without the intervention of a cashier or teller and thus eliminating the possibility of human error.

The peripheral can be used in one- or two-keyboard configurations, can be divided between two CPUs, and can thus be used by two tellers. It is therefore fitted with a double dispensing unit, which indicates to which teller the banknotes are to be supplied.

The driver reserves use of the peripheral to the programmer who first opened a work session. If another user wishes to use the peripheral he must wait for closure of the current session.

The programming interfaces provided by the driver allow sending commands and data and the reception of information on the outcome of the operation requested, or on the state of the peripheral. The commands are sent in parametric mode, so that the user does not need to know the form of commands expected by the peripheral. The driver offers two types of session: cash dispensing and servicing.

In the first type of session, the operator keys in the amount to be delivered to the client. The application program interprets the keyboard input and sends the unit driver an appropriate command for dispensing the money. The driver interprets the command and sends it to the unit, which then begins dispensing the cash.

The second type of session permits, for example, refilling the hoppers with banknotes, or carrying out diagnostics.

Among the commands that the driver is able to receive and interpret are the following:

- dynamic checking of the dimensions of single banknotes
- validity checks
- checks on serial numbers
- checks on the recipient of the banknotes (in the case of units shared between two tellers)

The driver programming interface is visible only to users of PASCAL+. To use it requires knowledge of the system identifier of the WS to which the peripheral is connected. For more details refer to the "Terminal Driver" section.

COBOL allows access to VISA badge reader/writer interfaces.

For the operation of the peripheral, the operator must be guided by program messages.

RS232/CL DRIVER

The RS232/CL driver handles the RS232 serial interface of the SIC or MUX controller, or CPU, and provides I/O interfaces at line level; these, unlike those of the WSM, are not specific to the peripherals to be connected. By means of the RS232 serial interface, which satisfies the CCITT V.24 international standard, it is possible to connect to an L1 MOS system any compatible external input or output equipment (for example: printers, measuring devices, paper tape readers and perforators, and graphics peripherals such as plotters and tablets) using links such as Current Loops, Modems and RS232s.

The RS232/CL driver permits execution of the following operations:

- Opening/closing a line, with the possible definition (in the first case) of transmission characteristics (for example: reception speed, transmission speed, protocol type, parity type, number of stop bits).

- Reading/writing of data on the line.

- Reading/updating of line states (for example: timeouts used, number of characters received).

- Suspension of the program while waiting for data from the line.

The programming languages Interpreted BASIC, Compiled BASIC, COBOL, FORTRAN, and PASCAL+ present the user with a set of functions (or instructions, for Interpreted BASIC) that interface with the RS232/CL driver. In this way, it is possible to execute whatever processes are required, and to define (in place of the defaults) the programming characteristics of the communications line through which the data exchange takes place between the L1 MOS system and the peripheral.

For further information on the RS232/CL driver programming interface, refer to the manual RS232/CL Interface, Programmer Guide.

ENCRYPTION DRIVER

The encryption driver permits encryption and decryption of data strings, and checks on the PIN (Personal Identification Number). The driver, in association with the badge reader/writer and PIN Pad, is typically used for recognition of the user, who inserts his personal magnetic badge in the badge reader, and enters his PIN via the keyboard.

The string received from the badge is decrypted, processed and compared with the number keyed in. Only if the two match can the user proceed to use the system.

The driver is based on the encryption and decryption subset of the ERU, which is an intelligent controller.

The driver interface is visible only to PASCAL+ users. For further information, refer to the manual PIN Check Driver Interface, User Guide.

The driver is also visible to COBOL users in the CAT handling context, as indicated in the manual CAT, CUSTOMER ACTIVATED TERMINAL, Application Interface, Programmer Guide.

SOFTWARE FOR HANDLING THE CAT WORK STATION

The CAT (Customer Activated Terminal) is a specialised work station of the SST (Self Service Terminal) line, and is part of the range of OLIVETTI products available for the banking market. It is connected to the system through ELB 1382.

It is installed in bank branches, or other controlled and protected internal premises, and allows the client to carry out a number of self-service transactions.

The functions offered by CAT are:

- obtaining the current balance
- transferring funds
- printing a bank statement.

The principal components of CAT, which allow exchange of information between the client and the system, are:

- The screen/keyboard, which guides the client in the choice and execution of the transaction. The keyboard is numeric and is fitted with acoustic feed-back. The screen is 9-inch and monochromatic.
- The (motorised) badge reader/writer, which enables the data to be read on the 1st and 2nd tracks of the credit card, and to be read and written on the 3rd track. The first and second tracks are defined by ISO standards as read-only tracks, and for this reason contain only fixed data, while the third track contains both fixed and variable data.
- Receipt printer (PR 2890), which allows program-defined printing of account statements. This is a dot-matrix printer with automatic cutter.
- ETB: a module enabling the return of the credit card when the line is down; it also provides an orderly closure of the current transaction.

Besides these basic components, the following modules present in the L1 system are used:

- ERU: a module allowing verification of the client's Personal Identification Number (PIN) after it has been entered via the keyboard.
- Real Time Clock: a module that makes available to the application the date and time at which the transaction was effected. This information may also be printed on the receipt, displayed on the screen or transmitted on line.

See also the section "Encryption Driver".

The CAT software programming interface is available in PASCAL+ and COBOL.

For further information on this subject refer to the manual CAT, CUSTOMER ACTIVATED TERMINAL, Application Interface, Programmer Guide.

SOFTWARE FOR HANDLING THE PC AS A WORK STATION

OLIVETTI Personal Computers can be used as L1 MOS work stations via the emulation programs L1WSE, WSELAN, or OLIEMU.

These emulation programs, run on the PC, allow at least some of the following operations:

- Emulation of an alphanumeric work station.
- Emulation of a graphics work station.
- Handling work station printers connected to PCs.
- Recording the screen contents or transmitting them to a printer.
- Handling of banking peripherals linked to PCs (PIN Pad, Badge Reader, CA2000, ML700/LP700).

The functional and operational differences between the three programs are described below.

L1WSE is used for "general purpose" functions. It only provides for using the PC as an local alphanumeric or graphics work station.

WSELAN is used for "general purpose" functions. It provides for using the PC as a OLILAN-networked alphanumeric or graphics work station, and handling a work station printer (IBM standard parallel interface, connected to the PC).

OLIEMU is used for banking functions. It provides for using the PC as a local alphanumeric or graphics work station, handling a work station printer (OLIVETTI standard serial interface, connected to the PC), and handling banking peripherals connected to the PC.

Emulator programs also allow the transfer of byte-stream files from MOS to MSDOS and vice-versa, using the TRANSF command present on the L1 MOS system, and also allow transfer of control between emulator program and standard MSDOS programs or commands, by means of the CONTSW program available on the PC.

The latter function (Context Switching) permits one activity to be run in the background and one in the foreground, leaving the background task suspended until control is returned to it.

By using the emulator programs it is also possible to delegate to the PC interface parts such as PGU, GSP, and VISA. Performing such operations directly on the PC saves memory and CPU time on the L1 MOS system.

6. HANDLING COMMUNICATIONS LINES

The CSS (Communications Subsystem) is the MOS subsystem that handles communications between MOS systems (distributed architecture), and between MOS and other systems, by means of communication lines.

The CSS is designed to make both MOS-to-MOS and MOS-to-foreign communications available to users in a straightforward and easily manageable form, which ensures that the following elements are transparent:

- The means of communication (lines and protocols)
- The location of the CSS user (application program) in the configuration of a distributed architecture

Such transparency is made possible by the CSS structure, which has the following characteristics:

- The CSS presents a single procedural interface to application programs. This interface remains unaltered when the topology of the underlying network and the whereabouts of the application program vary. It can be called by the MOS high-level languages COBOL, Compiled BASIC, and PASCAL+, and allows users to write sequential applications for data processing that can be transported from one system to another in the configuration.
- The CSS itself automatically handles all aspects of communications except data processing using user-defined parameters. The CSS functions are:
  - the making and breaking of connections to application programs on foreign systems, according to the line management configuration tables predefined by the user in a series of files
  - the transmission and reception of data on the physical lines whose characteristics have been predefined by the user in the system configuration file.
  - the management of the line protocols specified by the user in the system configuration file
  - the transfer of data within the MOS distributed architecture configured by the user on the local system and the systems connected to it.

The figure below shows how an application program (AP) can access line services regardless of its location in the MOS configuration.

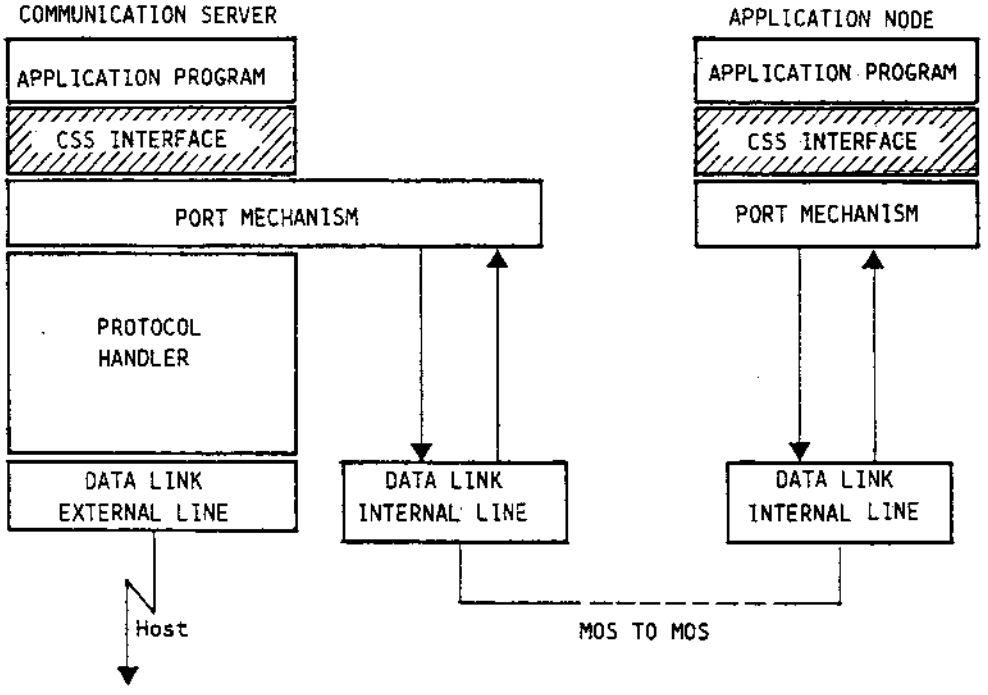


Fig. 6-1 Communications in a MOS Cluster

It can clearly be seen that the interface used by an AP on a client system to access line services is the same as that used by an AP resident on the communications server that provides the services. The way in which the two systems communicate with one another (via the port mechanism) is completely invisible to the APs.

The CSS can be used for implementing many different types of communications, as follows:

- Local area networks (LANs), comprising:
  - peer MOS systems interconnected via OMNINET or ETHERNET
  - hierarchical MOS systems interconnected in a star configuration (cluster) via the LION interface with the HDLC protocol
  - mixed LANs with MOS systems providing services for MOS and PC clients connected via OMNINET or ETHERNET or the RS232 interface (known as the OLILAN environment)
  - mixed or heterogeneous clusters handled by a MOS system by means of a primary protocol - 3270 primary, BSC3 primary, GOL11 primary or 2848 primary

- Wide area networks (WANs) interconnected via the private network architecture of IBM, SNA, or the public X.25-based packet-switching network, or the telephone network in conjunction with the LAPB protocol, or an X.21 circuit-switching network

- Lines to foreign hosts using a character-oriented protocol - BSC1/2, 3270 BSC3, 2848, GOL11, Burroughs B500/700, or VIP 7700/7800

- Concentrations of COSMOS or other non-homogeneous systems handled by means of an extended primary protocol implementation - COSMOS Master with BSC3 primary or the T5 node for SNA or BSC. In this case the CSS on the MOS server supplies its satellite systems with MOS services via a Bridge component to the MTS environment, and with connections to other foreign hosts via a Passthrough component.

The CSS services normally implement levels 1 and 2, and part of level 3 of the ISO/OSI reference model: that is, all physical and data link functions and some network functions.

In the case of systems in geographic networks (WAN), the communications services offered by CSS may be extended by the addition of the OLIVETTI product ONE (Open Network Environment), which implements ISO/OSI levels 4 and 5 (transport and session), and also supplies, optionally, services oriented to application problems (ISO/OSO levels 6 and 7: presentation/application).

When the CSS is specifically configured as a T2 node for SNA users, the system communications services can be extended to guarantee compatibility with IBM NPDA program products (Network Problem Determination Application) and TARA (Threshold Analysis and Remote Access) by the addition of the MOS network management services package, NEMOS, described in the Chapter "Networks".

For further details consult the CSS (Communications Subsystems) Features and Functions Manual.

7. NETWORKS

This chapter describes the various network topologies in which it is possible to connect L1 MOS systems.

DISTRIBUTED SYSTEM

The MOS operating system may be used to achieve a distributed system, in which a number of homogeneous L1 systems are physically linked together to form a topology:

- of locally distributed systems (LAN), based on a local Ethernet or Omninet network
- of distributed systems based on a LION interface and HDLC protocol (cluster).

Port

Ports provide a solution to the problem of communicating among MOS systems.

They permit the interchange (send-receive) of information units (messages) between processes either on the same system or on different systems belonging to a distributed configuration.

Use of the port mechanism is independent of the means of communication (HDLC, X25, ... protocols) used among the MOS systems.

An identifier is associated with each port, which identifies its name, its system and the object (service) to which the port is linked. Only the port name must be known in order to send a message to it: MOS will locate the system where the port resides and send the message.

Access to the ports is provided by a specific set of primitives.

Further details about the port mechanism can be found in the manual MOS Basic Architectural Concepts.

LAN

The interconnection mechanisms between the various systems in the network are created by the operating system and give complete and transparent resource, service and access interface distribution on all the network systems.

All the distributed systems in this topology have the same capacities: the resources connected to each system are available to all the others and managed globally as well as locally. Programs can be executed, and files accessed, regardless of which machine they reside on. This feature is due to the fact that each resource is uniquely identified by a path name that is independent of the system to which the resource is connected. A global catalogue is kepy of all the resources.

The main LAN characteristics can thus be summarised as:

- a single set of global distributed resources that can be accessed by all the systems in the network
- the ability to identify an element uniquely.

LOCAL NETWORK ENVIRONMENT SERVICES

The services in a local network environment are available through resource distribution and the standalone system's services.

There are two types of service:

- Remote access services for resources such as disks, lines, printers (spooler), and Batch monitor
- Distributed execution support services:
  - Name server for handling global network names
  - Route server for handling the system's configuration.

They have the following principal characteristics:

- The ability to identify a resource, regardless of its position in the network
- The maintenance of standalone systems in their correct configuration and working order. This ensures that applications are not altered when they are moved from a standalone system to a local network, or, within a local network, from one machine to another.

Several distributed systems can be defined on one LAN, where each one is characterised by its own set of resources. These resources are then unavailable to the other systems, unless the ONE services are used. An example of a distributed system in a local network is given in the figure below.

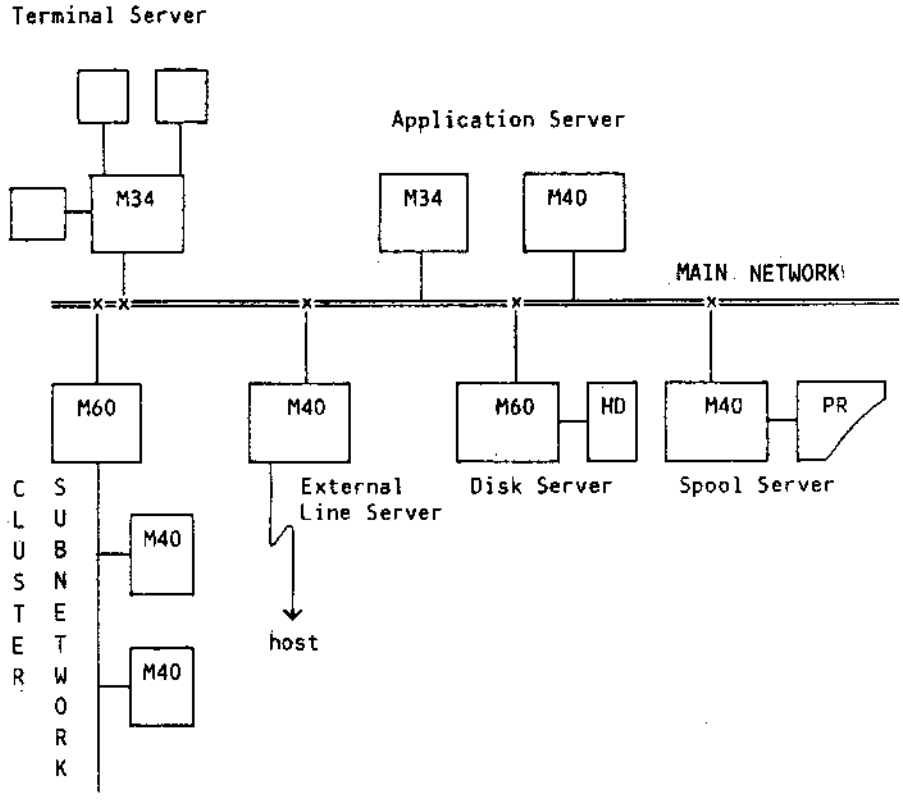


Fig. 7-1 Example of a Local Network

It can be seen from the figure that a system in the network can have one or more functions:

- It can act as a Server system, able to receive service requests from other machines in the network.

- It can act as a Client system, able to make requests to one or more Servers. This system is known as the Application/Terminal Server.

The figure also shows how a local network can be made up of a main network with internal subnetworks. A local network or a cluster are called subnetworks if they are connected to a main local network via one of their systems.

The connecting system is shared between the main or both distributed systems and, in turn, be accessed by both.

The subnetwork in the present figure is a cluster.

Remote File Access

The files physically resident on the various systems in a network are logically grouped into a single global file system.

The various file systems of each system in the network are nodes of the global file system, and are logically connected to it when the system is switched on.

The user can decide whether to export his files (that is, make them globally accessible). Two Shell commands are available for adding files to or deleting files from the public resource catalog (MKGLOB, RMGLOB). In order to identify the files, the user must indicate whether the one he wants to access belongs to the local file system (that is, it is resident on the machine being used - see the section on path names in Chapter 4), or whether the file is to be searched for in the global file system (that is, on all the machines in the local network).

The path name in this case has one of the following structures:

/../resource name

or

/../machine name/.....resource name

depending on whether the file is included in the distributed system's global resource catalog. If the file has not been exported, it can only be accessed if the user knows on which machine it resides.

CLUSTER CONFIGURATION

A cluster configuration is a distributed system made up of a "master" system and other "satellites" connected to the master via a communications line. Unlike the local network, the systems in the cluster do not all have the same capacities but are structured hierarchically. The master serves all the distributed system's resources, and supports the global services. The satellites act as clients of the master.

The master system handles:

- all the resources and services shared by all the cluster's work stations; for example, data files, communications lines to host, and queue manager.
- its own private resources.

The satellite system:

- can access the shared resources in the cluster
- handles its own private resources; for example, work stations and data files.

File access and file system object identification is described in the

section on local networks.

BACK UP SITUATION

Backup is possible in a distributed system.

Cluster configurations can include backup satellites to ensure continued functioning if the master system crashes.

The backup satellite normally carries out the activities of an ordinary satellite but it can also take over as the master by means of manual intervention by the user.

To activate the back-up satellite as the master, it is necessary simply to activate the switches that connect the master peripherals (disks and lines) to the back-up satellite, and then to carry out the back-up satellite IPL operations with the master software. During these operations it is not necessary to interrupt activity on the other satellites.

In local networks any one of the systems can act as the network administrator by means of operator intervention and the re-initialisation of the system. This system also maintains the preceding functions. The network continues to operate without requiring the re-initialisation of the other systems.

OLILAN

OLILAN (OLIVETTI Local Area Network) is a local network by means of which it is possible to interconnect OLIVETTI Personal Computers and Minicomputers of the L1 MOS line.

The advantages of such an environment are aimed at the users of Personal Computers: all the resources offered by the L1 MOS Servers are available to them as users of the network.

This means, in functional terms:

- **device sharing** : that is, the ability to share peripherals (usually disc units and printers) within the L1 MOS system network.

- **file sharing** : that is, the ability to access, subject to security restrictions, files resident on remote devices. In particular, OLILAN Remote File Access provides a mechanism for opening and locking/unlocking remote files identical to that implemented in MSDOS 3.1 by the Xenix primitives.

- **remote processing** : the ability to execute, from an MSDOS environment, a MOS program on an L1 MOS system.

- **terminal emulation** : the ability to use a Personal Computer exactly like any other L1 MOS terminal.

OLILAN Network Topology

A single level OLILAN network may be based on one of the following three types of transmission media:

- Omninet

- Ethernet

- RS232

In the case of Omninet or Ethernet cables, a "common bus" network topology will be used; in the case of RS232 cables, a star topology.

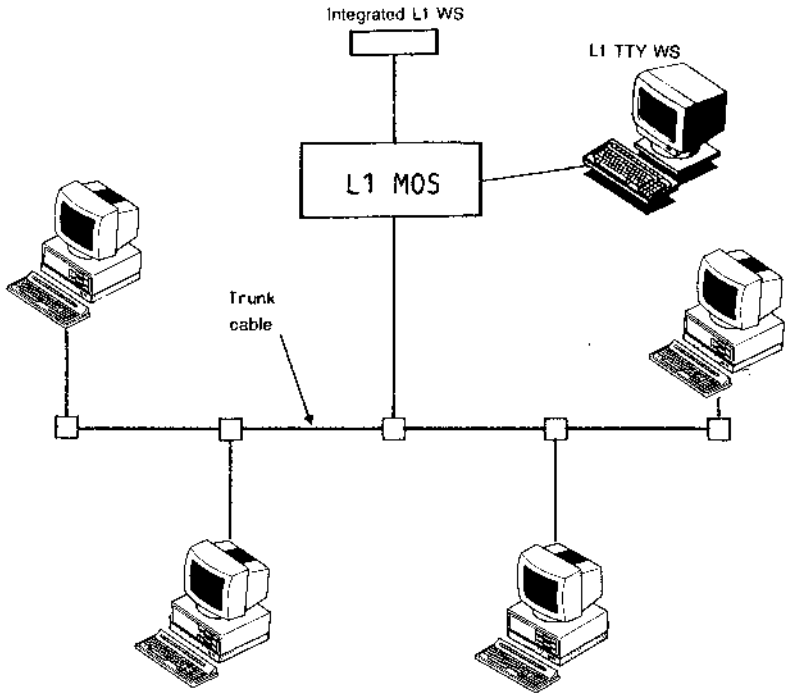


Fig. 7-2 OLILAN with Common Bus Topology

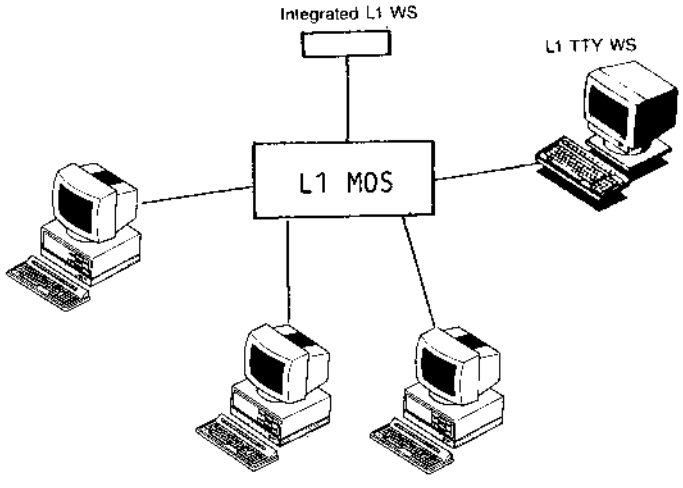


Fig. 7-3 OLILAN with Star Topology

In the case of a multilevel network (that is, a network composed of a series of sub-networks), the single sub-networks may use either Omninet or Ethernet cables. The presence of multiple levels is completely transparent to the PC user. The only limitation is the need for at least one configuration file per sub-network.

ONE

ONE (Open Network Environment), when applied to the L1 MOS system, simplifies communications between routines in closed systems connected across open systems.

By "open systems" is meant a local network situated among closed systems, whether L1 MOS or from another supplier, over which these may communicate by means of the standard protocol with which each is equipped.

A number of application routines currently available with ONE are listed below:

- FTF (File Transfer Facility). A reliable and flexible tool for transferring files between a local system and a remote system, and vice-versa.

- RPX (Remote Program Execution). Activates the execution of a program from a remote system.

- BNS (Batch Network Server). Permits the transfer of batch files through the network.

It is also possible to implement application programs through a series of interface directives that the user inserts in his own programs. These programs may be written both in PASCAL+ and COBOL.

Depending on the type of open system to which access is made through ONE, two cases are possible:

1. For local area networks Omninet and Ethernet, and the sub-networks HDLC (High Level Data Link) and X.25, the architecture of the ONE network is based on the ISO-OSI model (International Standards Organisation-Open Systems Interconnection), and access is made to it by means of the SLAM interface (Session Level Access Method).

   By using an X.25 network configured appropriately, it is possible to communicate with Honeywell-Bull DSA network hosts, in order to exchange data between programs resident on these remote systems and the L1 systems.

2. In IBM SNA networks, it is possible to access ONE through the SNAM interface, which provides a subset of services available for the SLAM user.

In the first case, ONE provides the functions of the following levels of the ISO-OSI models:

- Transport - This level implements "data transport" and renders the higher levels independent of the sub-network (which optimises transmission costs); to communicate with other routines it uses operating system messages and common data areas.

- Session - This level implements the ONE "user" services, handles communications between remote applications, and communicates with other processes using operating system messages and common data areas. It

accesses them by means of the SLAM interface.

Logically placed above these levels are the application processes themselves, which correspond, from the point of view of the functions, to the Presentation and Application levels of the ISO/OSI models.

In the case of the SNA network, the architecture of the ONE network is based on the MOS Line Manager directives. The same SLAM application routines are available also for SNA. The SLAM interface can be programmed in PASCAL+ and BASIC.

When used in an SNA network, L1 MOS systems with ONE are configured as IBM type 2 (T2) nodes, and communicate with each other by means of the PASSTHRU application resident on the IBM host (T5 node), which operates in the VTAM environment.

Further information on the ONE network software are contained in the manual Introduction to ONE.

NETWORK SERVICES

This section describes the services available in L1 MOS system networks.

NMS

NMS (Network Management Service) is the network service that OLIVETTI has designed for the management and control of the hardware and software resources of L1 MOS connected over a geographic network. This network may be either an X.25 network or an SNA network over which L1 MOS systems communicate using the ONE Session level (SLAM interface for the X.25 network and SNAM interface for the SNA network).

NMS offers alarm control and processing via the Alarm Handling facility, and software distribution over the network via the Software Distribution facility.

NMS is organised in two main modules, CMS and LMS, which may reside on one system alone, or be distributed on a number of machines:

CMS Central Management System is a module residing on the NCC (Network Control Center, L1 MOS system) that controls the entire geographical network, and has the following functions:

- Alarm Handling: processing, recovery, and archiving of the alarms sent by the controlled network systems.

- Software Distribution: distribution of software (including programs and data files) over networks, in order to guarantee that the same software version is available at all times on all the networked systems.

LMS Local Management System (Agent Component) is the module residing on every controlled system and which:

- cooperates with the corresponding CMS, sending it the alarms generated by the L1 MOS system resources and "collected" by the LMS Server system module (described in Chapter 13 "System Maintenance") - Alarm Handling

- makes available, on the chosen date, the software that has been distributed - Software Distribution

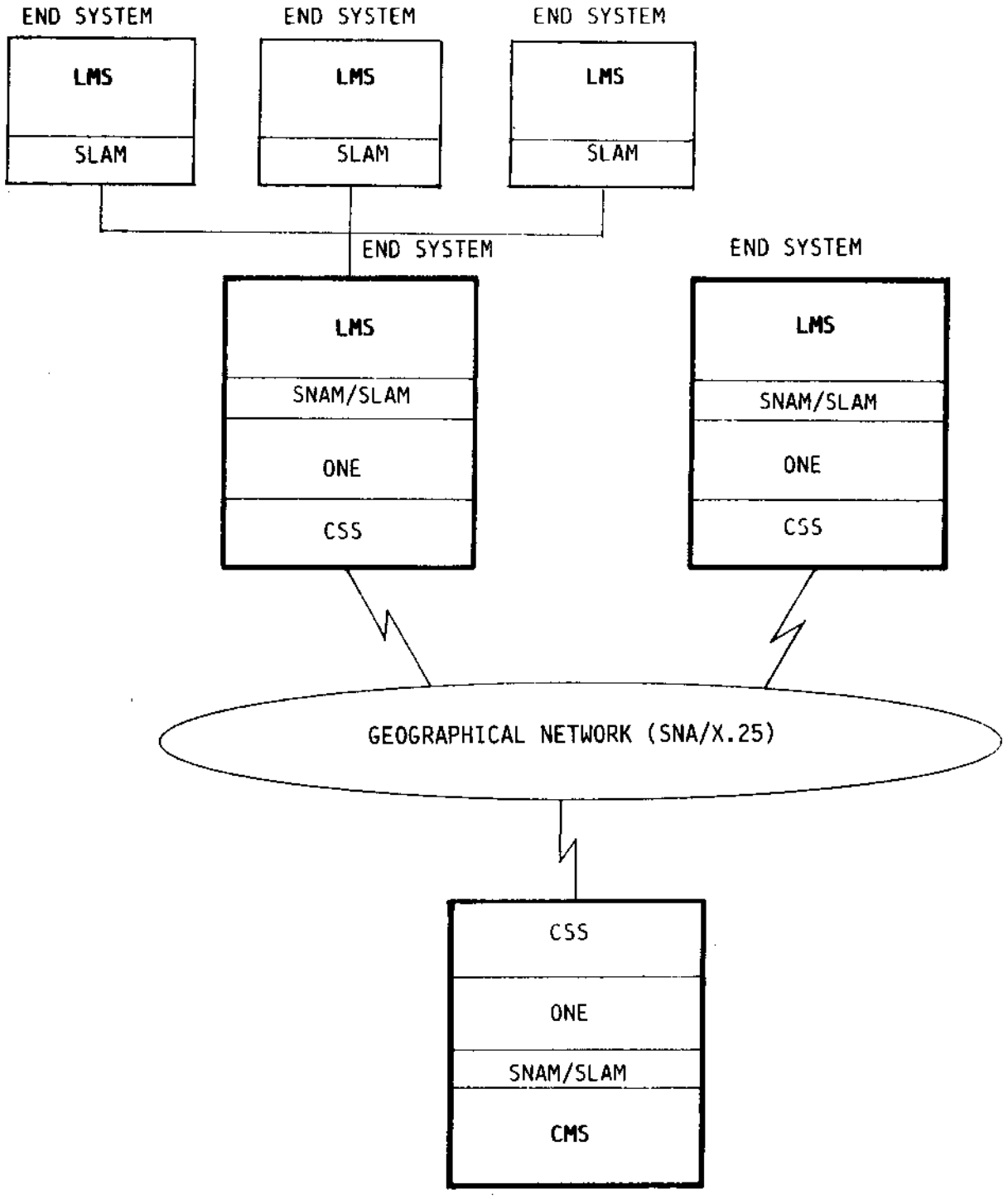


Fig. 7-4 Network Management System

The figure shows the NMS structure (controlled systems are indicated as End System).

An extremely important feature of NMS is its configurability: the user can configure either one or both of the NMS functions.

The Functions

The functions offered by NMS are Alarm Handling and Software Distribution. Each consists of one part that is executed on every system under control, and another part that is executed on the NCC (Network Control Center), as shown in the figures below. The operator has a simple interactive interface that allows him to handle the alarms (display them, request technical assistance, store them in archives), and to prepare and distribute the software.

Alarm Handling

An alarm is a message generated on an End System for signalling a hardware or software error condition, or some important system event.

Alarm Handling (AH) is the NMS function that handles alarms. The components of AH that reside on LMS (LMS Agent) send the alarms to the CMS, where the recovery is handled. LMS Agent must respond to the requests of the network management services (NMS), using the tools provided by the LMS Server component of the controlled system. The LMS Server component is described in the "System Maintenance" chapter.

From the system point of view, the LMS Agent component can be regarded as a set of application programs, resident in the MOS user environment, that carry out various functions and access the Server component by means of system interface primitives. The basic functions carried out by LMS Agent are:

1. To check the alarm channel that the Server provides, by means of a system primitive, and to send alarms to NMS, using the MOS Communications Sub-system (CSS) features.

2. To send alarms to the CMS for recovery. The way in which the alarms are sent to the CMS can be configured. The alarms may be sent:

   a) in real time

   b) after a certain condition has occurred (that is to say, the user may define a filter)

It is advisable to install LMS also on the NCC, which, in this way, can keep itself under control.

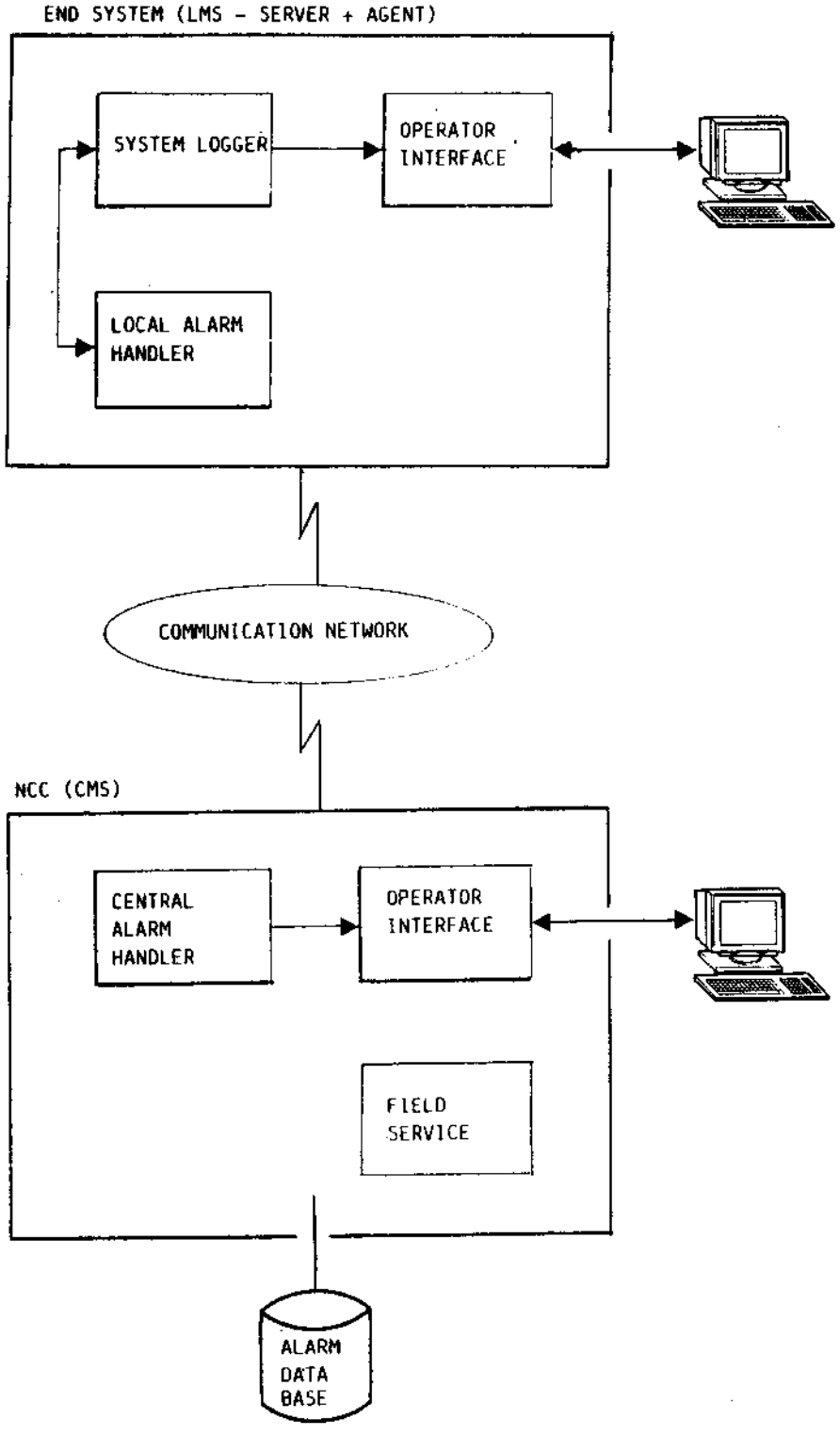


Fig. 7-5 Alarm Handling

Software Distribution

The NMS Software Distribution (SWD) function enables transmission of "software elements" (which are objects of the MOS File System) from the CMS to the LMS. For transmission over the network the ONE File Transfer feature is used.

The operations necessary for the distribution of the software are started and controlled by the NCC operator and carried out:

a) partly on the CMS - preparation of the distribution, that is, of the volume containing the "software elements" to send, and the destination End Systems

b) partly on the LMS - copying, if the network is local, to all the End Systems involved and activating the software on the chosen date.

"Activating" means making available the new software on the same date on all the End Systems involved, in order to guarantee consistency. The checking operations are carried out on the NCC.

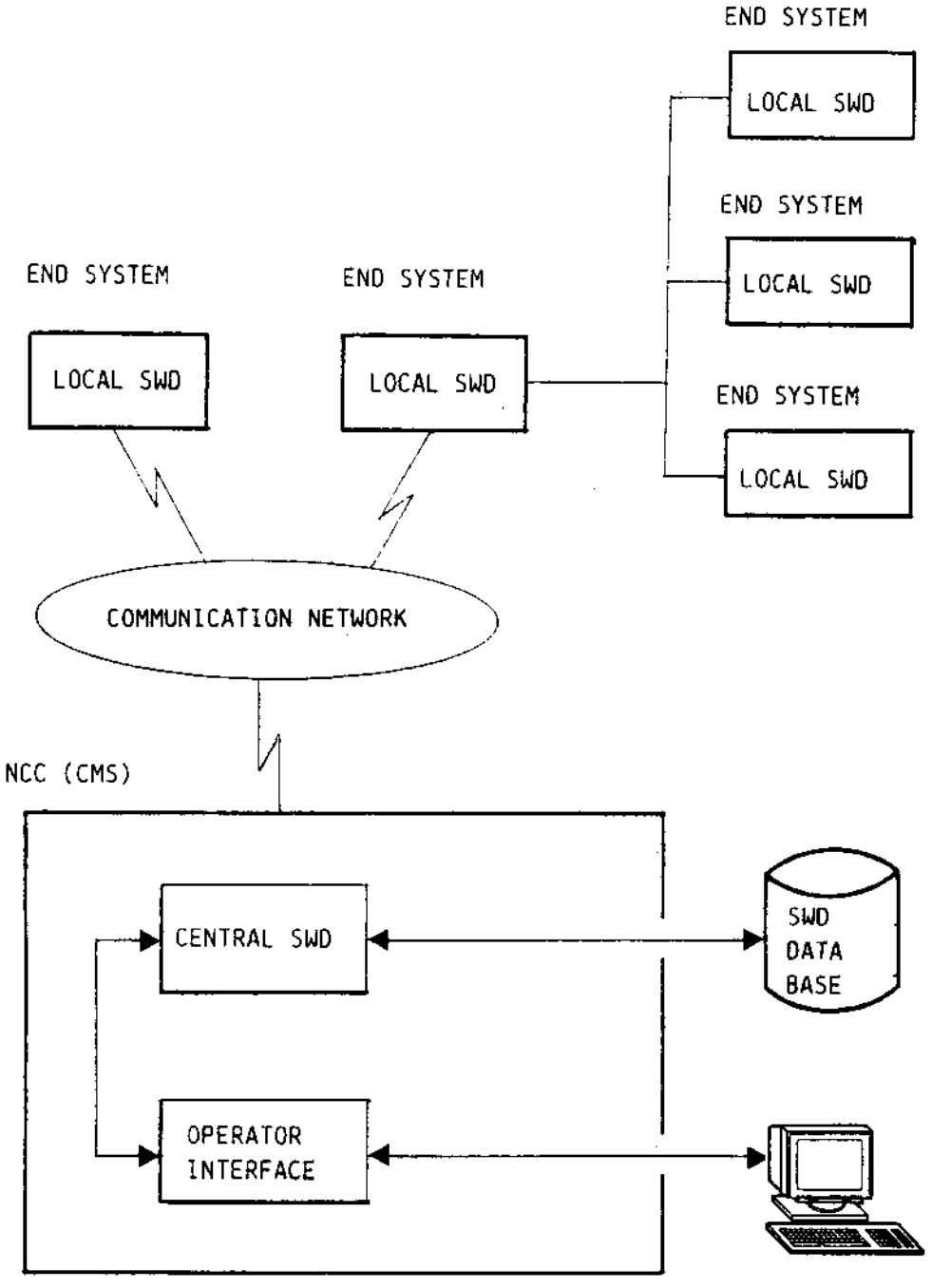


Fig. 7-6 Software Distribution

NEMOS

NEMOS is the product that OLIVETTI supplies for making IBM monitoring possible on L1 MOS systems in an SNA network. Thanks to NEMOS, the IBM host "sees" the L1 MOS system as if it were an IBM system; in fact compatibility is offered with IBM products:

- NPDA program (Network Problem Determination Application)

- TARA (Thresholds Analysis and Remote Access)

using two agents, Alarm Agent and End to End Agent: both may be customised.

NEMOS permits handling:

- RECFMS 00, 01, 02, 03, and 05 for NPDA support

- RECFMS 04 (data dependent on PU/LU) for providing compatibility with TARA.

The three main components of NEMOS are: CNMS (the server that supports the exchange of RECFMS/REQMS records), Alarm Agent and End to End Agent. The figure below illustrates the components of NEMOS, the programs produced that reside on the host, and the records handled.

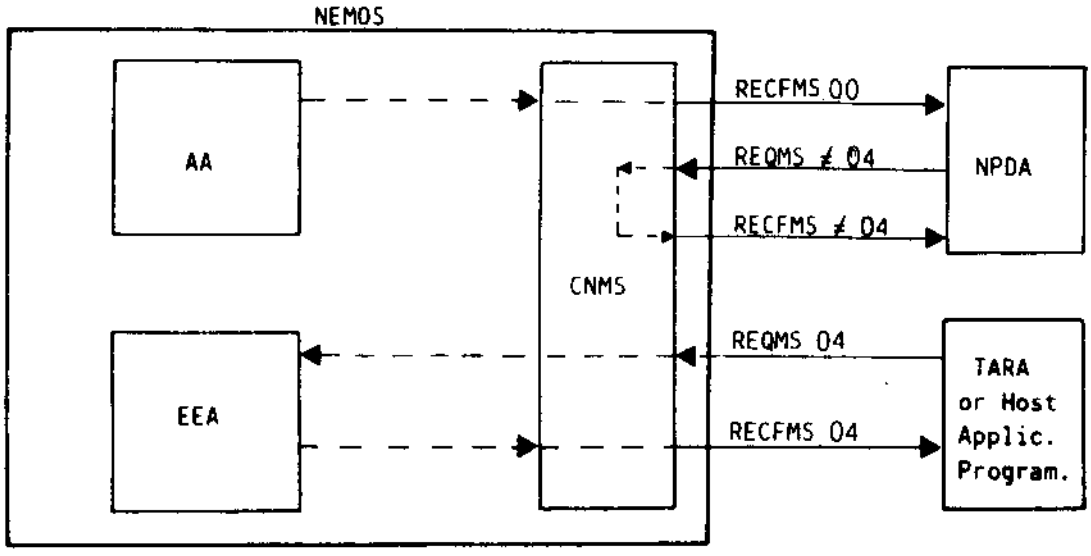


Fig. 7-7 NEMOS

Communication Network Management Service (CNMS) interacts through the CSS with the network handling programs resident on the IBM host, and also with the two agents Alarm Agent and End to End Agent.

The **Alarm Agent (AA)** receives the alarms generated by LMS Server by means of a CSS primitive, provides all the information necessary for generating an alarm message in the RECFMS 00, calculates the appropriate recovery action for the alarm in question, and then passes all the relevant information to the CNMS, which sends it to the host.

The **End to End Agent (EEA)** interprets and executes the requests sent by the network operator (on the host) and decoded by CNMS. EEA passes the requests to the MOS component responsible (Shell, PMM, etc.) using the primitives available on the interface.

PART 2 - APPLICATION PROGRAMMING

INTRODUCTION TO PART 2

This part of the manual outlines the main characteristics of the available high level languages, including methods of program preparation and execution. The system packages, application environments, and tools available for program development are also presented.

For further information about the available languages, see the MOS Programming Languages, Features and Functions manual.

8. PROGRAMMING LANGUAGES

The MOS operating system supports two interactive programming environments: a COBOL environment, known as ICE (Interactive COBOL Environment) and an Interpreted BASIC environment.

The system also provides the FORTRAN, COBOL, and Compiled BASIC languages for application programming.

The multifunctional characteristics of MOS mean that all the available languages can be used, and so programs written in different languages can be executed, simultaneously, from different work stations or sequentially from the same one.

Application programs (in different environments) executed from different work stations can exchange information via concurrent access to shared files (provided that file protection is guaranteed by the languages used).

Application programs (in different environments) executed sequentially from one work station can exchange information via access to shared files (provided that the type of file used is handled by the languages of these environments).

The programming environment described below can be activated in either the MCL or BEAM (Business Environment Application Monitor) environment.

COBOL PROGRAMMING ENVIRONMENT

Introduction

COBOL (COmmon Business Oriented Language) is a high level programming language that has been developed particularly for commercial use.

It was initially developed in 1959 by the main American computer manufacturers and certain government bodies, and rapidly became the most widely used programming language in the world.

OLIVETTI L1 MOS systems use the COBOL and ICE-COBOL languages.

ICE-COBOL

ICE-COBOL (Interactive Cobol Environment-COBOL) has an integrated software base that can create an interactive COBOL environment at all levels.

The ICE-COBOL environment has the following components:

- CRECOS (CREator of CObol Sources), which allows ICE-COBOL language source code to be generated interactively.
- The compiler, which translates ICE-COBOL source code into intermediate code.
- ICRTS, ICE-COBOL's run-time system, which provides the user with an interactive program preparation environment, using one of two menu systems:
  a) The ICE Logon Menus, which are intended for general program development.
  b) The ICE MONITOR Menus, which are a set of menus with built-in security and priority features intended for a business environment.
- Standard utility programs, which mostly convert files and programs coming from other systems.
- A set of system utilities, which allow operations such as printing spooled files, redefining the ICRTS parameters, and exiting from the COBOL environment (shutdown).

Tools are also provided for transferring ICE-COBOL programs from the OLIVETTI S6000 systems to L1 systems.

COBOL

The COBOL supplied for L1 MOS systems implements the features of ANSI 1974 COBOL to level 2, as well as many features of the ANSI 1983 Recommendations. The Screen Section, an OLIVETTI extension, is also implemented.

Programming in the COBOL environment is supported by the following components:

- The COBOL Compiler, which checks the source program's syntax after an error has been found. This ensures that as many statements as possible are checked for syntax in a single step. The compiler produces an object code file for the linker and an Internal Symbol Dictionary for use in debugging.

- The OLINK linker, which has a set of commands for linking object code files, creating modules with overlay structures and including the VISA, PGU, MTS, and Line Monitor interface libraries.

- The COBOL run time routines, which are held in a shared load module to ensure that only one instance of these routines is loaded into memory.

- The Symbolic Debugger, which uses the Internal Symbol Dictionary produced by the compiler to permit program debugging using symbolic addresses.

The application program can be activated under MCL, BEAM, or MTS, or directly by Grandpa.

BASIC LANGUAGE PROGRAMMING ENVIRONMENT

BASIC (Beginners All-purpose Symbolic Instruction Code) is a high level, general purpose language used in commercial and scientific applications. The OLIVETTI L1 MOS system uses Interpreted BASIC and Compiled BASIC.

INTERPRETED BASIC

The Interpreted BASIC language consists of an interactive environment for program preparation and execution.

It has a simple user interface, which allows easy control of the BASIC environment.

It is used to solve both commercial and scientific problems not involving a large number of calculations - for example, controlling experiments.

BASIC source programs can be created and edited with both the System Editor and the BASIC Environment Editor.

When the program has been prepared, it is submitted to the interpreter.

This provides features for program debugging such as, for example, displaying and/or modifying variable contents, starting and stopping program execution, and activating the "trace" facility.

A BASIC program's execution can be speeded up by "compiling" it before it is saved. This permits faster access to variables.

Once the "compiled" program has been saved, however, editing, debugging and listing operations cannot be carried out.

BASIC provides other features besides those defined in the standard programming language. These include a graphic interface, allowing the use of PGU graphics (see Chapter 9), and the RS232/CL interface, which allows any serial peripheral to be accessed by a BASIC program (see Chapter 5).

BASIC can access positional and keyed files as well as sequential files.

COMPILED BASIC

The Compiled BASIC language, available on the L1 MOS systems, conforms to the standards proposed by the American National Standards Institute (ANSI: BASIC ANSI X3J2/83).

Compiled BASIC has been designed for large, complex applications requiring high speed execution. Program development and maintenance is simplified by the use of the most widely known programming language (BASIC) and by the high level programming features offered for modularising complex programs.

Source programs in Compiled BASIC are written using the system Editor.

COMBAS

The source program is compiled by the "COMBAS" compiler to produce the object modules. These are subsequently linked.

A BASIC program is made up of a main module and zero or more additional modules (or none). Each module can be individually compiled, linked, and tested.

Once these modules have been compiled they can contain subprogram calls (individually compiled) and can access the various packages and system interfaces via CALL (VISA, PGU, GSP, RS232/CL, Line Manager).

OLINK

The compiled modules, the subprograms and the system packages must be linked together by the "OLINK" linker. This creates a single executable module, constituting the final application program.

A BASIC program can be executed from the Shell environment, from BEAM, or directly from Grandpa after the IPL phase.

Exception-Handler

Compiled BASIC also offers the "exception-handler" feature that "traps" run-time errors: recovery operations can be carried out, or errors can be handled in the normal way, within a program.

External Files

Compiled BASIC provides sophisticated file handling. Files in the compiled BASIC environment can be organised in four different ways:

- Sequential: a sequence of records. Only sequential access is allowed.

- Byte stream: a sequence of bytes without any grouping criteria. Only sequential access is allowed.

- Random: a sequence of records, identified by number. Direct access is allowed.

- Keyed: a sequence of records identified by keys. Direct access by key is allowed. Keyed files can have an external key or an internal primary key, and up to five internal secondary keys.

Lock

Files, and even individual records in random and keyed files, can be protected.

I/O

I/O statements give powerful screen/printer handling and check the input fields.

Functions

A large number of both numeric and string system functions are available, and make the Compiled BASIC language very versatile.

Scientific and commercial applications can be carried out and precise results obtained.

Numeric and string functions can be defined by the user.

A program must be structured if compiled modules, subprograms, system packages and functions are used, therefore Compiled BASIC includes logical and control structures not usually present in BASIC.

PROGRAMMING IN THE FORTRAN LANGUAGE

The FORTRAN language available on the L1 MOS systems is the standard FORTRAN 77 with some extensions added by OLIVETTI.

It is particularly suited to solving technical-scientific problems, and can use the graphic functions and the RS232/CL interface functions offered by MOS.

PROGRAM PREPARATION AND EXECUTION

The source programs are created using the system Editor and compiled by the FORTRAN compiler (which can be activated in the BEAM or Shell environment).

The compiled units can contain calls to subprograms that have been separately compiled, and must be linked to the necessary libraries (run-time, I/O, mathematical, and, optionally, graphic and RS232/CL interface) with the "ZLOC" linker.

A program prepared in this way can be activated in the Shell or BEAM environment, or automatically after IPL.

FILE HANDLING IN FORTRAN

A file consists of a sequence of records. There are three types of record: formatted, unformatted, and end-of-file.

File records must be either all formatted or all unformatted, with the exception of the last one, which can be an end-of-file record.

Each open file has a pointer that indicates the current position in the file. The file access method depends on the type of connection to the file, and defines the way in which records can be read or written.

Files with fixed or relative organisation can be sequentially or directly accessed. Sequential files can only be connected for sequential access.

Internal files can also be handled. These allow data to be transferred from memory to memory, instead of between memory and files. The operations allowed on internal files are read and write, formatted or unformatted, with both sequential and direct access.

9. SYSTEM PACKAGES AND SERVICES

MOS supports a number of packages. This Chapter is a general introduction to them, and describes:

- VISA: a screen formatting package
- PGU: a graphics package, oriented to CAD/CAM applications
- GSP: a graphics package, oriented to commercial applications
- Signature verification: a system service for checking signatures, oriented to BC application
- COMMIT: a system service for safeguarding data integrity
- Message Switching: a system service for exchanging messages
- A set of other services offering application users various functions.

VISA

The VISA package consists of two programs, VISA and TFORM, that allow logical work station handling.

VISA handles the data files that an application program:

- sends to the work station's peripherals for displaying and printing
- receives from the work station's peripherals, after read operations or keyboard input (VISA handles peripherals such as the screen, keyboard, printer, badge and magnetic stripe reader).

To carry out these operations, VISA loads into memory (from a library stored on disk) and interprets the form indicated by the application program.

A format is a sequence of logically organised fields, each of which can contain an input value to (or output value from) an application program.

The form contains, for each field, the definition of the field's characteristics (field attributes). For example:

- field name
- input field
- peripheral giving the field value
- field length
- numeric field, with sign
- field to undergo validity control, etc.

The field attributes are used by VISA to handle the data read and write operations correctly.

The TFORM utility program is used to:

- create and catalog the forms in the corresponding library on disk. Creating a form means defining the fields of which it consists and providing the values of the field attributes for each of them.
- handle the library, modify, delete, copy, display, and list the forms.

Form Execution

The application program uses particular routines to execute a form, which constitute the VISA interface. VISA stores and interprets the form from the library on disk, sending the output data to the appropriate peripheral. The data are provided by the application program. VISA then receives the input values provided by the operator.

If the value of certain fields has to be checked before being accepted, VISA executes a tabular control for each of them, using a control table defined in the form, and/or passes control to the appropriate validity routine inserted in the application program, according to the specifications in the field attributes.

When the form has been executed, the application program receives the input data from VISA, originating from the designated peripherals.

The VISA package has the following characteristics:

- It runs both on L1 MOS systems and on OLIVETTI PCs.

- It can be used by application programs written in COBOL, Compiled BASIC, Interpreted BASIC, DMS, and PASCAL+.

- It handles "sub-forms": that is, particular groups of fields within the form.

- The routines inserted in the applications program for using VISA can act:

  . on the whole form

  . on a sub-form

  . on a single field.

- The application is not suspended while the peripherals are reading the data to be transferred to it by VISA.

- It can be used in single- or multi-process mode. In the latter case, multiple forms may be executed simultaneously by a single application program.

- Different application programs may use the same form simultaneously; in this case, only one copy of the form exists in memory, which is shared by the various programs (only the data areas are private to each program).

- The read and write peripherals are defined as:

  . write unit: at form and field level

  . read unit: at field level.

  This information, given when creating a form with TFORM, can be modified by the application program when the form is executed by VISA. Read and Write modules can be defined at subform level.

- Checks (if any) on the field values are carried out by a validation program that is logically linked to the form, but exists as a separate entity and is not included in the application program. The validation program is written in a "BASIC-like" language known as VPL (Validation Processing Language).

Greater detail on the VISA package is given in the manual VISA, Form Handling Package, Features and Functions.

PGU

The L1 MOS PGU graphics management package offers a set of features for developing two dimensional graphic applications. It is a general-purpose graphics package that provides routines for line drawings and "raster" manipulation routines. The functions and procedures of this package may be called from the FORTRAN, COBOL, PASCAL+, Compiled BASIC and Interpreted BASIC programming languages, and by the Data Management System (DMS).

The PGU is available on L1 MOS systems. Using OLIVETTI Personal Computers as work stations on an L1 system, it is possible to carry out part of the graphics processing on the PC itself, thus reducing the work load of the L1 System CPU.

The PGU graphics management package may be used to write graphics application programs ranging from the simple to the most highly interactive. It is an integrated package of more than 100 Pascal+ subroutines intended for such applications as CAD/CAM, process simulation, advertising, cartography, etc.

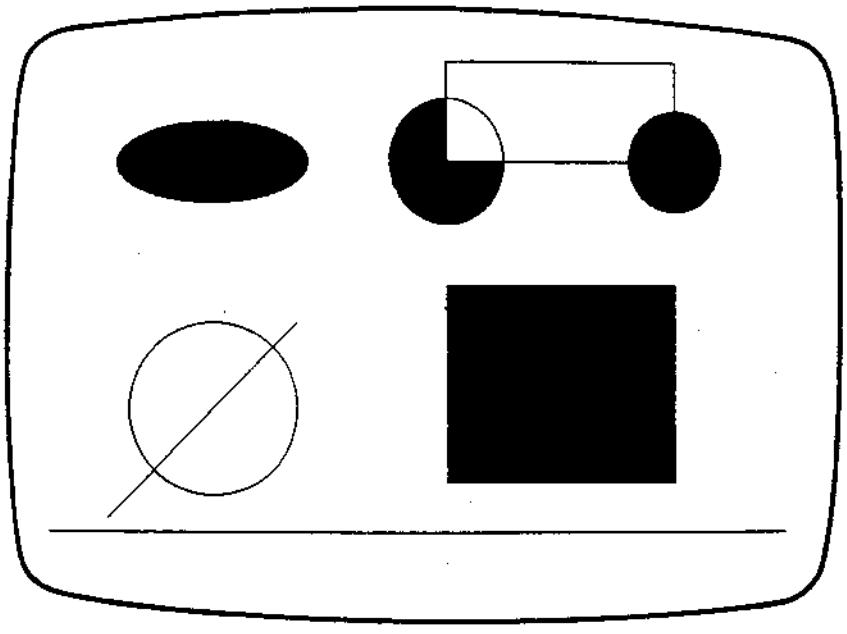
The PGU package combines the advanced architecture of the microprocessor with "master-scan" display technology. The colour and monochromatic graphics videos include auxiliary memory for displaying pictures with pixel-by-pixel control of the image.

The user interacts with the computer through the standard terminal keyboard and/or one or more tablets, which are seen as graphic input work stations.

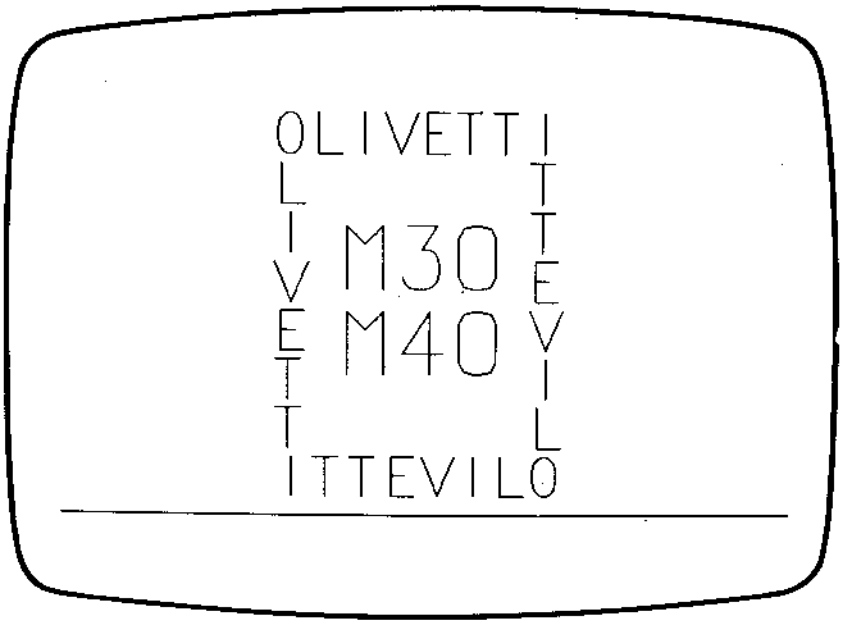
The functional capabilities of the PGU graphics package may be subdivided into fourteen general classes, as follows:

- Control: these functions include opening and closing the graphics session, enabling and disabling buffers, setting the PGU to the graphics or alphanumeric state, and so on.

- Output: these functions can display pixels, text strings, lines, rectangles, circles, ellipses, and sections of circles and ellipses. Closed shapes may be filled with one of the available colours (8 on a colour screen, 2 on a monochromatic screen).



- Attributes: the output colour, the logic operator, line style, filling pattern, character orientation, and many other attributes may be set by the PGU.



GSP

The Graphing Subroutine Package is a collection of high level graphics subroutines that allows graphs to be created by programs written in the PASCAL+, COBOL, BASIC, and Interpreted BASIC languages. The results depend on the design supplied by the programmer and the data supplied by the user at run time.

The output may be in colour or monochrome, and may consist of pie or bar charts, or plots, and may include text.

Attributes

The GSP has three attributes:

1. It is easy to use. The user can create a graph on the screen quickly and with a minimum of commands. The user who has created a graphable data base, containing the x and y coordinates of a sequence of points, may then have the graph automatically formatted and displayed.

2. It is flexible. The user is able to override the design decisions made by the system by changing the value of design variables, such as colours and labelling standards. In short, the user is able to design a unique graph. The GSP gives this freedom by permitting the adjustment of any of the variables that the GSP consults in formatting a graph.

3. It is easy to automate. The user can construct graphs with special design characteristics without having to specify these design characteristics each time a new graph is created. The system's default design instructions can be replaced with "personal" default characteristics so that, subsequently, the default graph format need not be overridden. This is done by saving the "personal" format specifications in a data file that is recalled when creating new graphs. A library of standard format files can be constructed, and application programs producing graphs with these formats can then be written with a minimum of programming effort.

Getting Started

The L1 MOS Editor (or an application program) is used to prepare the data to be graphed. This should be at least one set of x and y coordinates (and perhaps some labels and titles). A graph is then created to the default design. The programmer may accept this graph as it is, or rearrange the design.

What and How

The decisions to be made in constructing a graph consist of two groups or sets. They are the primary (or basic) decisions, concerning what will be graphed, and the secondary (or design) decisions concerning how it will be graphed. A basic decision is one that affects the information

conveyed by the graph. All other decisions are secondary.

The information resulting from basic decisions is the "primary data base" or "graphable data base" of a graph. The information resulting from secondary decisions is the "secondary data base", "format data base", or "design data base" of the graph.

The Primary Data Base

The user constructs a primary data base, locks it, and makes no further alterations to the primary data of the graph. The information in the primary data base is then used by the system to construct a secondary data base according to default values. The user may call immediately for a graph to be displayed, or first modify the design of the graph by adjusting the design variables.

The Secondary Data Base

The GSP contains a hierarchy of design variables. The default values that may be adjusted are relevant to:

- control and layout
- axes and plots (individual and collective)
- pie charts and text
- rectangles.

Less significant variables are re-adjusted when more significant variables in the hierarchy have been altered. This occurs automatically unless the dependence is switched off.

Disk Storage

The primary or secondary data bases of a graph, or the graph itself, may be stored on, and subsequently retrieved from, disk. This means that once a set of design decisions have been made, and have produced an acceptable graph, the design can be restored automatically for subsequent graphs.

Application Interface to GSP

The application interface to GSP produces graphic output for two types of applications:

1. The addition of graphics to a task-oriented program, such as a general ledger or current accounts package. The user might be a data processing specialist in a large bank. A program that automatically reviews accounts every month might generate graphics.

2. As a basis for interactive graphics packages. With such packages, the user constructs presentation quality graphs by working repeatedly at a keyboard; one format may be tried, modified slightly, viewed, altered again, and so on until satisfactory. The initial user of the GSP might be a professional programmer designing an interactive package that groups together sets of GSP functions, adding procedures and interface menus so that the end-user can construct graphs easily.

The application interface is integrated with the PGU (Graphics Management Package). The GSP user can use all the functions of the PGU without the PGU being explicitly opened or closed. Moreover, the GSP and PGU share the same font files, and the text font files "cheapie.00" and "cheapie.01". One function of the PGU permits the division of the physical screen into windows, so that a number of graphs may be activated simultaneously.

The GSP is not visible to the end user of the application program.

As with the PGU, using OLIVETTI Personal Computers as work stations of an L1 system, it is possible to carry out part of the graphics processing on the PC itself, thus reducing the workload on the L1 system CPU.

THE SIGNATURE VERIFICATION SYSTEM SERVICE

MOS provides two PASCAL+ routines that permit signatures to be verified by using the PGU graphics package. One special routine allows the signature on a piece of paper to be obtained in Modified Hoffman Code via an optical reader (scanner), and the other enables it to be reformatted into a bit map, in normal or zoom format. The signature may be subsequently displayed using the PGU "PUT" routine, and stored.

THE COMMIT SYSTEM SERVICE

MOS safeguards data integrity using the commit mechanism. If a user request to update the data base is blocked, the operation is repeated as though the request was being made for the first time.

Certain functions (available with PASCAL+, COBOL and Compiled BASIC) enable the user to define the Commit Regions: these consist of a set of distinct operations, which are logically grouped to carry out the user's request.

For example, the user is guaranteed that a program will either update two files, or neither, if an error is found in the Commit Region.

The system is responsible for handling the necessary structures, which are:

- the Stage Area, a part of memory that stores write operations, requested by the application program and interpreted by the Commit service
- the Log file, where the situations before and after each operation are stored
- the Dual Log file, which duplicates the data in the Log file in case of damage to the storage medium
- the transaction file, where the identifier of the last series of correctly executed operations is stored
- the JOUCMAN file, which keeps track of the operations carried out, and is accessible using the DISPJOUC utility.

These structures enable the Commit service to restore the last correct situation automatically when non-fatal errors occur, so that the user can carry on working.

The RECOVERY utility is available for restoring the last correct situation after a fatal error has occurred (for example, a damaged disk).

MESSAGE SWITCHING

MOS permits, in a distributed configuration, the exchanging of messages with applications running on systems external to the configuration.

For this purpose PASCAL+ routines are supplied that allow the exchange of "immediate messages" and "non-immediate messages" between users of the distributed configuration. These routines may also be called by a COBOL program.

"Immediate messages" are messages sent asynchronously from one user to another and immediately displayed on the asynchronous window of the consignee's terminal.

"Non-immediate messages" are messages not immediately displayed on the consignee's terminal, but queued in a "mailbox" to be read at the consignee's request. It is possible to create a mailbox for each machine at every system start-up, or to append to the existing ones.

The messages sent by a user are inserted in the mailbox to which the consignee belongs; an active selection routine concatenates the messages for each user and updates the pointers. The chain of messages is initialised with the first message addressed to that user.

After a user has read a message, the pointers of the message chain are updated, and the message is logically, but not physically, deleted from the mailbox.

VARIOUS SERVICES

Listed below are a number of other services that MOS offers the application programmer.

USER DUMP

Following abortion of an application program, if the user has previously enabled the primitives "Enable Dump", "Disable Dump", and "Memory Dump", described in the manual MOS Programmer Guide, he may use the DMPRINT utility, which interprets and prints a UASD file (User Address Space Dump).

ADDING TO THE LOG FILE

The routine WRITEUSRLOG can be called by COBOL and PASCAL+. It enables application programs to record errors or events occurring during program execution (for example, start and end time of a activity), or to add a record to the Log file. There are two writing modes: normal and extended, depending on what is specified in the Grandpa configuration file. The normal mode enables writing a record only if some of their fields have a certain value; the extended mode writes the record in all cases.

TABLES

The TABLE_GRID function, which can be called by both COBOL and Compiled BASIC, simplifies the creation of grids and tables by using program-defined characters.

LOGOFF

The LOGOFF function, which can be called by COBOL and PASCAL+, suspends the connection of a switched line between a remote work station and the L1 MOS system, enabling savings to be made on telephone line charges.

EXEC

The EXEC function calls and executes an MCL activity (such as an executable program or MCL routine) from a PASCAL+ program running under Shell or Grandpa, returning control to the calling program on termination of the called activity.

10. SOFTWARE ENVIRONMENTS

The software environment defines and controls the user's operating environment. MOS constitutes the basic support for different environments oriented towards specific application sectors.

This Chapter describes the general characteristics of some of these software environments, namely:

- MTS
- BEAM
- DMS
- Terminal Emulators

MTS

MTS (Modular Transactional System) is a modular software environment that provides the user with tools for creating transactional applications, where maximum programming flexibility is required for creating user transactions.

MTS allows a homogeneous subsystem to be defined, in which multiple users use the same set of application programs, calling on a shared set of transactional services and working on a shared database. MTS software guarantees recovery of updates to the files handled so that the data correspond to the transactions executed.

Besides MTS, a transactional system also includes:

- work stations
- data files
- application programs.

The files handled by MTS are standard MOS files, plus a file type specific to MTS, the Chained Data Base (CDB).

The CDB file is a data base integrated into the MTS software, which has the same structural characteristics as the reticular model. Its main constructs are files and sets (relationships between files).

Each set is characterised by an owner record and a member record. The conceptual scheme logically describes the structure of files and sets. The external scheme represents the user's view of the data; it may be the same as the conceptual scheme, or a subset of it. A user can organise an application under MTS using:

- Interactive Programs (IP)
- Server Programs (SP)
- Background Programs (BP).

COBOL and PASCAL+ are the two programming languages that can be used for writing these types of programs.

Interactive Programs (IP) and the execution of user transactions.

Server Programs (SP) called as subprograms by the Interactive and Background Programs.

Background Programs (BP) activities. These functions also require Transactional Services (SP) but can be executed without operator interaction.

Standard MTS software transactional services can be called as normal subprograms from Interactive, Server, and Background Programs.

These standard services, as well as guaranteeing file handling, also provide other functions such as:

- message switching between the transactional system's work stations (this service is also available in the SHELL environment)
- interactive program switching
- dynamic reconfiguration of the transactional system
- checking the functioning of the internal line
- collecting data on the transactional activity.

MTS provides a utility program that helps the user to configure all the elements of a transactional application (work stations, files, application programs). Thus, MTS can be adapted to meet the needs of any installation, optimising the system's functioning.

Further information on the features of MTS is given in the manual MTS Features and Functions.

BEAM

The BEAM software environment (Business Environment Application Monitor) handles the requirements of L1 MOS commercial systems, providing the following functions:

- Simple and customised interfaces
- Flexible resource protection mechanisms
- Control of conflicting activities
- Inherent system services for executing environment activities and for preparing and controlling the environment
- Multifunctionality

The BEAM environment facilities can be used by MOS in both distributed and non-distributed environments.

Ease of Use and Customisation

BEAM is user-friendly and can be customised.

Operator intervention is only requested when necessary; activities, programs, and procedures are selected from hierarchical and customised menus.

Authorised users may receive information on the parameters and connections of the activity; other users use a much simpler system.

The operator's job is also simplified by messages and help menus.

All the environment's interfaces can be configured, and the national language may be used.

Resource Protection

BEAM checks that each user is authorised to use the resources he requests. More precisely, each user can:

- use only certain work stations
- from these, select only those activities that both he and the work station are allowed to access
- modify the activity's execution environment only if permitted, and only in the way permitted to him.

BEAM's resource protection mechanism is flexible and can be used for:

- single- and multi-sector businesses

- businesses with their own processing centre and those using central control.

BEAM resources can be both customised and shared between users of the same business sector and/or those with a specific hierarchical position in a company.

Control of Conflicting Activities

The BEAM environment allows a file or another structure that is shared by several activities to be temporarily locked by one of the activities.

Environment Services and Programming Languages

Programs and procedures can be activated under BEAM, and programs can be written in any of the languages available in the MOS environment. Procedures, however, can only be written in MCL.

The Shell environment, and COBOL ICE, BASIC, and DMS interpreters, are activated by BEAM without operator intervention.

BEAM activities can use the MOS services of the batch and interactive environments, and can also use the spooler. BEAM also includes an inquiry service.

All the MOS commands can be configured as BEAM activities, thus allowing BEAM preparation and control operations to be carried out within the BEAM environment.

Controllability and Reliability

BEAM has a Log file in which certain system events are recorded automatically. Others are recorded only if this is requested at configuration time.

BEAM also provides the necessary interfaces through which applications may record significant events in the Log file.

The environment includes both BEAM and MOS services for hot, warm, and cold restarts on information and activities.

Multifunctionality

BEAM can either be activated from any work station or only from specific ones, depending on Grandpa's configuration. It can use the whole range of MOS work-stations, both intelligent and non-intelligent, with the exception of VT100 and VT100-like terminals.

The BEAM environment can be either the only environment, or one of several on a work station. If the latter is true it can be activated as an alternative to:

- the office automation environment
- the terminal emulation environment
- the Shell "general purpose" environment
- any user program.

These alternatives can also be integrated into the BEAM environment: more precisely, they can be seen as a BEAM activity and so use its features.

For further details on the BEAM software environment features, see the manual BEAM Features and Functions.

DMS

DMS (Data Management System) is a software environment that provides the user with a language with which to access MOS files. It provides the following operations:

- File creation and updating
- Enquiry
- Report production

DMS:

- provides each user with a personalised view of the data in the (virtual) file by using particular descriptors
- minimises redundant data and software by using the same physical data structure to define different virtual files
- allows the user to break away from the conventions of traditional programming, thus facilitating the writing of application programs for commercial use
- protects the files from unauthorised access.

The available data handling language is called DML (Data Manipulation Language), and is distinguished by ease and flexibility of use. It may be used:

- interactively: each DMS command entered is immediately executed
- to execute a command file, which is a sequence of DMS commands stored in a file
- to create and execute a procedure, which differs from a command file in that it allows the definition of symbolic parameters to which the user assigns an actual value when the procedure is executed.

The physical file descriptors mentioned above are created by the Data Administrator, using the Data Definition Language (DDL). These descriptors are stored in a special file known as the Data Dictionary, which is protected by the DMS against unauthorised access.

DMS is oriented towards file handling, report generation, and graphics, and, in these application areas, is proposed as an alternative to the traditional programming languages.

DMS commands exist for:

- opening and closing files
- creating, modifying, and deleting records
- selectively displaying record fields

- producing graphical output
- producing printer output
- generating reports with selective conditions and data operations
- dynamically relating two or more files
- activating and using programs outside the DMS
- using VISA forms for the input and representation of data at the work station.

As well as handling positional and keyed files, the DMS works on byte-stream files, and may use the Commit facilities.

The following utilities are written in DML and can therefore be run in the DMS environment:

- **DMS Multifile/Conversion:** provides, in addition to the Multifile features, the option of converting positional or keyed files (MOS environment) into byte-stream files (PC environment) and vice-versa, thus simplifying the exchange of data between the two environments.
- **DMS Query By Form:** provides interactive access to the main DMS facilities. Does not require any specialised knowledge or the writing of programs.

Further information on DMS features can be found in the manual DMS, Data Management System, Features and Functions.

TERMINAL EMULATORS

Terminal emulators constitute a software environment providing the facilities offered by certain IBM terminals, by emulating their functions. The emulated terminals are:

batch: the 2780 and 3780 terminals, emulated by the OBF package (On-line Batch Facility)

interactive: terminals 3278 and 3279 of the 3270 class, emulated by the SNA 3270 Emulator package; and 3776 terminals, models 1/2 and 3/4, of the 3770 class, emulated (in the SNA environment) by the RJE 3770 Emulator package.

All Emulator types use the services of the MOS Communications Subsystem (CSS).

2780/3780 EMULATION

This type of emulator permits files to be sent and received in file transfer and remote job entry modes between an application on the host system and an L1 MOS system.

The functions provided may be logically divided into:

on-line: communication line reception/transmission control, and control of the emulator's global activities (for example, handling the queue of files to be transmitted, selecting output devices to receive the files)

off-line: file insertion/removal in the transmission/reception queues, preparation of files to be transmitted (job or command files), and manipulation of files received.

On-line activities must be executed on a specific work station, defined as the "controller" at configuration time.

Off-line activities can be executed (in the Shell environment) on any other work station at the same time as other user activities, independently of the on-line activity.

The off-line processing of files to be transmitted, and files that have been received, significantly reduces the use of the communication line to the host.

On-line reception/transmission activities can be executed in unattended mode (that is, without the operator's interactive control).

SNA/BSC 3270 EMULATION

The functions offered by these two emulators enable the user of an L1 MOS work station to integrate with an application program being executed on IBM host, in SNA or BSC network architecture, as if it were running on an IBM interactive terminal of type 3278 or 3279.

The application on the host displays on the L1 MOS work station a number of masks containing information and/or data requests. The emulator checks and validates the data entered at each level, checking, for example, that a field defined in a mask as numeric (by means of special attributes) has been given a numeric value, and so on. On completion of all the checking operations, the emulator sends the validated data to the application on the host.

Emulation of the 3270 system in SNA and BSC environments consists, in fact, of the emulation of various products of the IBM 3270 series, that is:

- the cluster controllers 3274 and 3276 in stand-alone or master-satellite configurations
- the 3278 video terminals (black and white video, model 2) and the 3279 (four-colour video, model 2)
- the 3278 and 3279 printers, DSC or SCS (the latter type only for SNA).

3770 RJE EMULATION

This emulation package allows L1 MOS systems to initiate Remote Job Entry (RJE) activities, interacting with an application program resident on an IBM host in the SNA network architecture, emulating the functions of the interactive terminals IBM 3776 models 1/2 and 3/4.

The 3770 RJE emulator normally uses diskettes in place of the punched cards used by the IBM 3770 series terminals.

Further information on terminal emulators is contained in the manual CSS, Communication SubSystem, Features and Functions.

11. PROGRAM DEVELOPMENT TOOLS

This chapter introduces the program development tools available under MOS.

EDITOR

The standard Editor under MOS is a full-screen editor that treats the screen as one or more windows that may be opened on to one or more files to be edited.

GENERAL CHARACTERISTICS

The Editor allows the creation and updating of source program and text files. It may operate on various data types and may carry out specific operations on each type:

- for characters:
  - . insert
  - . replace
  - . delete
  - . append
- for strings:
  - . search
  - . search and replace
- for lines
  - . insert
  - . delete
  - . move
  - . join
  - . split
  - . append

    . add line numbers (display only)

- for the screen window:

    . open and close

    . select the active window

    . vertical and horizontal scroll

  The Editor also offers other interesting functions, which may be summarised as follows:

- large-file handling
- multiple-file handling
- more than one back-up level
- tabulation and justification

Multiple File Handling

The Editor allows editing sessions on more than one file at a time. This is done by logically dividing the screen into a maximum of 3 windows that display parts of the same or other files.

Each window may be opened or closed at any moment in the session and may be moved up and down within the text. At any given moment editing operations are permitted on the contents of the active window.

Back-up Levels

The Editor provides various back-up levels against loss of text (or textual corrections) caused, for example, by power failure. These operations can be requested without closing the current editing session.

The user may save the text on which he is working (with the appropriate command) and then carry on with his editing session. He may also request an update to the file without losing the previous version.

Another form of back-up is available, in which deleted text lines are stored temporarily, so that the user may recover them if he has deleted them by mistake.

Tabulation

Tabulation may be used when editing a text that has fixed fields.

Justification

Justification may be used to align a text between two or more margins predefined with the tabulation function. This alignment is obtained by transferring words from one line to the next and varying the number of spaces between the words.

OPERATING CHARACTERISTICS

After the Editor has been invoked from the Shell environment, a wide rectangle appears on screen, framing the whole area to be edited.

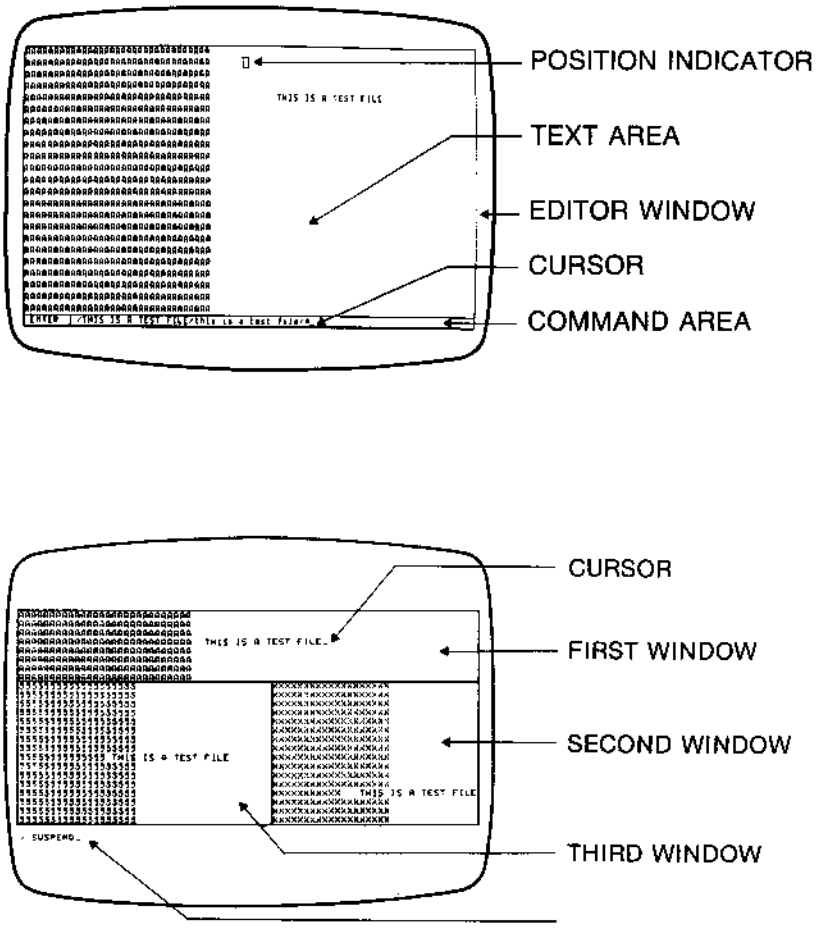


Fig. 11-1 Screen Format in the Editor Environment

The Screen

The screen is logically divided into:

the text area:	this contains the text that is to be edited, and can be subdivided into more than one window.
the command area:	this is used to give commands and to display the Editor messages. Since this area occupies the last line of the text area, each time it is used its contents are saved and then restored.
the system line:	this is the last screen line, which is reserved for system use.

The Keyboard

In addition to the keys commonly found on typewriter keyboards, the keyboard has a range of special keys. These are numeric keys, cursor movement keys, and a set of function keys used for entering the Editor commands.

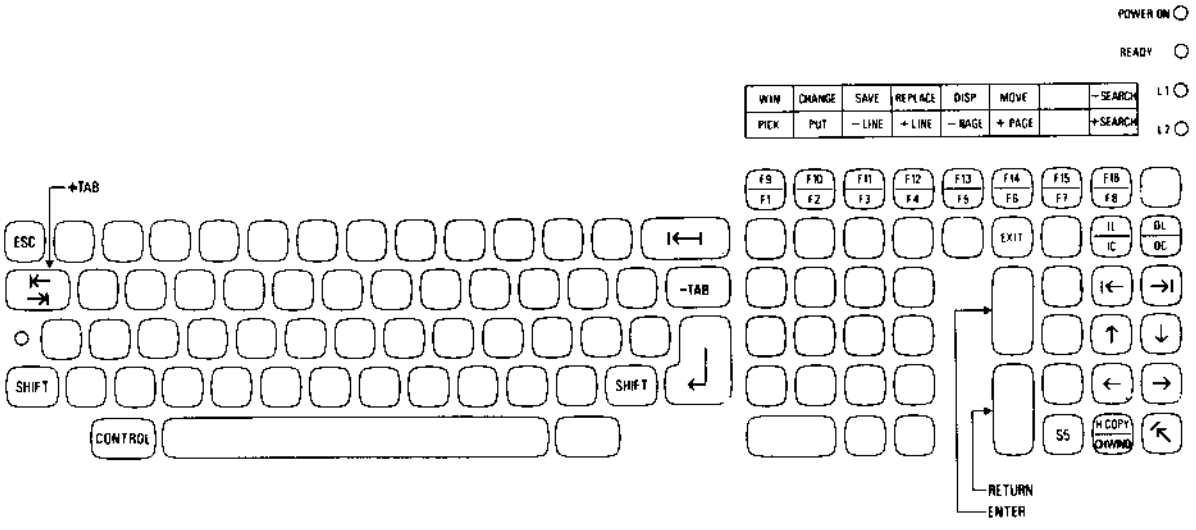


Fig. 11-2 Special Keys

COMMANDS

From an operational point of view the commands used for Editor control can be divided into:

- intrinsic commands
- simple commands
- compound commands.

The intrinsic commands (for example, character or line appending and character replacement) do not need a special function key: the cursor has merely to be moved to the correct position and the characters keyed in.

The simple commands are activated by pressing a single key.

For compound commands, after pressing **ENTER**, **EXIT**, or **DISP**, the parameters (if any) are entered into the command area, and then the function key that corresponds to the command itself must be pressed.

GENERALISED LINKER (OLINK)

The MOS generalised linker (OLINK) produces modules having a format suitable for loading into central memory, starting from the output from the BASIC, COBOL, FORTRAN, or PASCAL+ compiler.

GENERAL FUNCTIONS AND CHARACTERISTICS

The main functions carried out by the linker when creating a loadable module are the following:

- Allocating virtual memory with logical addresses. The program to be loaded in memory is split into several parts: code, data, constants, stack, and so on.
- Creating program segments for producing an overlay structure.
- Resolving references to external symbols. The use of external symbols means that several modules can use the same data or code.
- Checking compatibility between the definitions of symbols and the references to them.
- Generating lists and maps.
- Creating libraries (using the BUILD program).

OVERLAY

The linker allows the user to define an overlay structure for the generated virtual memory segments. This structure optimises memory use, grouping code sections with similar functions into separate overlay modules. Each overlay is loaded into memory when requested by the program, and may be overwritten when no longer required. Using this technique, the address space is conceptually enlarged.

OLINK OUTPUT

The output from OLINK is represented by a "program directory"; that is, a directory containing a file "main" (the main program), the overlays (if any), and the other files created by the linker.

COMMANDS

The generalised linker is called in the Shell or BEAM environment by specifying the name "OLINK", followed by a list of commands. These allow the user to:

- allocate an address to a program section within a memory segment
- specify the modules to be included in the program

- specify the libraries and loadable modules to be searched for or included
- specify the program's overlay structure
- specify the desired output (maps, listing, etc.)
- control the generation of the Internal Symbols Dictionary
- control the generation of the Entry Point List file. This file contains the entry point definitions and the data of a loadable module to which another loadable module can refer in a successive link stage.

DEFAULT FILE

There is a command file, known as the default file, that is read at the start of a program linking phase. It makes linking very easy for the user, as he only has to specify the object files making up the program.

UTILITY PROGRAMS

The **CHECK** utility is available for the production of statistics on object modules. It produces statistical information on the object modules specified, and makes a syntactic analysis of an OLINK command file. This utility is particularly useful during the creation of very large programs, as it can help the programmer to contain the program within the limits established for the various development tools.

The **MSLDUMP** utility reads an l-module and places on the standard output the information contained in it.

GENERALISED SYMBOLIC DEBUGGER

The generalised symbolic debugger allows complete interactive control of the execution of compiled and linked programs.

By using the debugger, the user can identify and remove program errors that cannot be detected by the compiler.

GENERAL FUNCTIONS AND CHARACTERISTICS

The debugger is interactive: the user follows program execution along with the debugger, altering and examining the state of the program as he wishes.

The debugger is symbolic: the program positions can be referred to by their symbolic names as they appear in the source program.

The debugger is generalised: it supports two programming languages, BASIC and COBOL.

The main functions of the debugger are to:

- examine or change the contents of the memory area where the user program is loaded
- obtain information referring to the symbols contained in the program, so that the same ones can be used in the debugging phase.

COMMANDS

The debugger is activated in the Shell, BEAM, or MTS environment and used interactively by means of a command language.

The main commands allow the user to:

- insert or remove breakpoints in the program
- display areas of memory, variables, and registers
- display the current program position and relative line number, state of execution, labels, and procedure names at each phase of debugging
- specify, if not working interactively, a file containing debugger commands and a file to which output may be sent
- modify an address or the contents of a variable or register
- execute a specific number of program lines
- interrupt and restart program execution.

PROGRAMMING TOOLS ON OTHER SYSTEMS

Programming tools for L1 applications are provided on the S6000 system.

ICE-COBOL CROSS-COMPILER

A cross-compiler for ICE-COBOL is provided on S6000, producing intermediate code to be run in the L1 ICRTS environment. The source code accepted by the cross-compiler is identical to that accepted by the L1 compiler.

FILE TRANSFER UTILITIES

Two utility programs (FFT) are provided to transfer programs or data from S6000 to L1 systems.

The first utility, which runs on S6000, may be used to transfer files from S6000 to floppy disk.

The second utility, which runs on L1, may be used to transfer files from floppy disk to L1.

In order to improve the portability of commercial applications, the functions offered by the VISA package on the S6000 system are guaranteed on L1 by an optional extension to the COBOL run-time support.

PART 3 - TOOLS FOR SYSTEM SPECIALISTS

INTRODUCTION TO PART 3

This part of the manual describes the support tools for system specialists who have to generate, install, and maintain the system.

It also describes the system programming language, PASCAL+, which provides a direct interface with the components of the MOS operating system, allowing it to be expanded, modified, or adapted to meet specific needs.

12. SYSTEM GENERATION AND INSTALLATION

SYSTEM GENERATION

System generation takes place in the OLIVETTI subsidiary centres, using a suitable L1 machine and a "starter" operating system.

The input required at the generation phase comprises the release archives, containing all the software modules available for a given release.

The output of the generation phase is the system software for a particular system, ready to be installed in the client's premises. It can be supplied on diskette or on magnetic tape, depending on the client's system configuration.

To ensure simple and correct system configuration, the modules of the release archive are represented logically in various alternative functional sets. A functional set, which can be that of the system (including the system modules), or of one of the application environments (including environment or language software), is the smallest unit of software that can be ordered by a customer.

FSU

The entire generation process of a client's system is supported by the Field Support Utilities (FSU) contained in the starter operating system.

These utilities are:

MAKECRTS to configure the type of COBOL run-time software

SYSCONF to set or modify the general configuration parameters, and create the configuration file $CON, the 1-module $IPL, and, optionally, the SYS volume.

LINECONF to set or modify the configuration parameters necessary for distributed or on-line systems, and to generate the Communication Subsystem ($CHM) module.

SYSGEN to create the SYS volume, containing the modules of the client's operating system and the configuration parameters file.

This self-explanatory utility guides the operator, by means of a series of menus, in selecting the optional or alternative modules.

The procedure then creates a system volume (SYS), into which it copies the chosen system modules from the release archives. It also asks the operator to indicate a configuration file previously created by means of SYSCONF, which is copied to the SYS volume. This file will be used at the IPL phase for initialising system modules.

DPCGEN to create the DPC volume, containing the languages and the environment software chosen for the client's system.

For every application environment or language available, the user may choose from a number of alternative functional sets.

At this point the generation of the system volume on disk is complete, and the distribution package for the installation of the system in the client's premises can be prepared.

The preparation of the user's distribution package requires the use of special commands, additional to the normal Shell environment commands. These are:

MKBOOT copies a bootstrap program from one disk to another. It must be used after the MKVOL command (or MKENV and MKVOL in the case of a diskette) before generating a system disk.

MKSYS prepares a tape (SCT or MT) with a bootstrap program and then copies onto it a SYS volume of a system generated on hard disk. This must be used for installing on systems without diskette drives.

The system volume may be copied from hard disk to diskette or tape, depending on the client's hardware configuration. During this phase the user login mechanism is activated, and the components for batch and/or spooling activities are installed, if required.

SYSTEM INSTALLATION

For systems with only diskette units, no installation phase is necessary: the system is loaded directly from the distribution diskette. For systems with hard disk, the software must first be transferred from tape or diskette onto the client's hard disk.

When the system is activated, the software is loaded and initialised using the IPL program. The system administrator may then set up volume and directory structures, and enter user names.

It is essential to divide the disk into volumes in order to be able to execute back-up operations.

The special commands to be used at the installation phase are:

INSTALLQM This includes the batch and/or spooler system in the configuration of an operating system generated on disk. It should be used only if these features are required.

PARSER — This produces a MOS user environment configuration file, to be used by Grandpa on activation of the system.

This command, with a suitably modified source file, may later be used for modifying the configuration characteristics before reactivating the system.

MKLOGIN — Activates the user login mechanism. Must be used during the preparation phase of the client's distribution kit.

SPCONF — This includes the spooling system in a configured operating system.

LINKDEV — Connects two or more hard disk units to form a single logical peripheral.

Further details on the above commands, and on how to install the operating system, are to be found in the manuals System Software Generation and Installation, User Guide and CSS Generation and Configuration, User Guide.

THE GENERATION, CONFIGURATION, AND INSTALLATION OF COMMERCIAL SYSTEMS

As an alternative to the generation, configuration, and installation chain described above, a tool is also available (**SYSCONFR**) that carries out all the actions necessary to complete these tasks.

SYSCONFR is particularly well suited to commercial systems, and can generate a system starting out from a reference configuration (to be selected from six possibilities), and modifying it on the basis of the hardware (work stations, disk drives, and so on) and software characteristics (application environments) of the client's system.

The system thus generated is consistent in its various modules ($IPL, $CON, SYS volume, and so on) and complete. It includes the Grandpa configuration file $CONFGP, the DPC volume, a starter operating system for initialising the client's system, and a routine to load the software thus generated automatically on to the client's hard disk.

MODIFYING CONFIGURED SYSTEMS

In order to avoid going through the whole cycle of generation, configuration, and installation for a partial modification of the system software (such as when adding or removing a work station or other peripheral), a special tool is available (**CONFMAN** , CONFiguration MANager) that allows the parameter values of the $CON file to be modified, under guidance, in order to satisfy the client's latest requirements.

In the case of minor modifications the Shell command **MODCON** can be used for routine modification of parameter values.

The Shell command **REBUILD** is also available, which enables the last configuration session (SYSCON and/or LINECONF) to be repeated in non-interactive mode. This will generate a system identical to the one

previously configured without requiring the interactive part of the configuration phase.

Further details on the above commands are included in the manual System Software Generation and Installation, User Guide.

13. SYSTEM MAINTENANCE

This chapter provides a general description of the MOS software maintenance tools, including:

- the Local Management System (LMS)
- software maintenance commands and utilities
- the system debugger.

THE LOCAL MANAGEMENT SYSTEM (LMS)

The term Local Management System (LMS) refers to the facility available in the MOS environment that is used to collect and record information relating to the logical and physical resources of the system.

This collection process is primarily intended to allow MOS to take part in an overall scheme of network management and control, such as that available in the IBM SNA or the OLIVETTI ONE network, but it also allows the MOS user to monitor the status of his own system and to perform certain maintenance tasks. In this context, LMS is the mechanism that provides maintenance statistics and alarm signals that are transferred to the Network Control Centre.

At an introductory level, the LMS may be regarded as consisting of two main components, the Server and the Agent. These work together to provide overall management services, but have quite different logical connotations:

- The Server is associated with the system as the local mechanism in the process of data collection and logging.
- The Agent is associated with the external network management service, to which it is responsible for forwarding the requested maintenance statistics and alarms.

This logical separation of LMS functions leads also to the possibility of a physical separation of the two LMS components, thus allowing a single Agent component to act on behalf of several Server components in a cluster or local area network, sending the collected maintenance statistics and alarms to the network management control centre.

Data input to the LMS come from the MOS resource managers, which are software components responsible for the total activity of a software or hardware resource (Process and Memory Management, File System, peripheral drivers, COBOL, Communications Subsystem, and so on).

At the initialisation of the system, a table (the Resource Map Table) is

created on the Server, into which the resource managers insert information on all the hardware resouces of the system.

In general, the resource managers supply statistical data relating to the activity of the resource. These data are recorded in a Log file on the Server, and can include:

- hardware, software, or operator errors
- events occurring
- the current values of the resource's activity counters.

THE SERVER COMPONENT

The LMS Server component is located on each system in the configuration, where it performs two basic functions:

- The collection of statistics and error messages received from the resource managers, and the maintenance of the system's Log file.
- Examination of the error messages and counters at specified intervals, and, if necessary, the generation of alarms.

System Logging

When the logging process is enabled, all relevant events occurring in the system are collected from the resource managers. The server decides:

1. whether to generate an alarm
2. whether to log the record.

Because LMS is accessed directly by the system, and also by the user environment (via the system primitives), two types of interface are provided for the logging mechanism. A further primitive is available for logging application data and can be called from either COBOL or PASCAL+.

Alarm Generation

The Server provides a mechanism for generating alarms when the incoming log record from a resource manager signals some particular condition, such as:

- an error reported in the operation of a hardware resource
- anomalies reported in the system software or in an operator function
- some aspect of system behaviour that must be reported to the Network Management Services (NMS)
- exceeding a threshold value set (using the utility DATACONF) for certain counters containing statistical data.

Alarm Routing

The alarms generated by the Server can be transferred to the Agent component of LMS, to be passed on to an external control centre and/or to an L1 Master Work Station (MWS).

The destination of the alarm is decided at the system configuration stage.

Tracing

The LMS compiles a trace file with information concerning the execution of the sets of system calls defined in the Grandpa configuration file, and selected using the TRACEMON utility. The trace file can be examined by the user with the REPTRA utility.

User Interface

The user interface to LMS consists of a set of system utilities, available to the MCL (MOS Command Language) user. These are application programs, and use the system interface extension (made up of primitives) in order to access the Server.

These utilities will:

- create or modify (even at run time) a Data Collection configuration file (DATACONF)
- transfer data collected in the system's Log file (DATAC)
- read from the system's Log file (REPLOG)
- interpret the contents of the trace file (REPTRA)
- save, modify, and read a record in the database of system messages (SYSERM)
- trace the system calls made by an application program (TRACEMON).

Database of System Messages

This database is composed of one record for each possible system message. Each record contains supplementary information on the message itself and on the suggested action to be taken.

The database is handled by means of the SYSERM utility (allowing, for example, the contents of a record to be translated into a national language), and its contents may be displayed by the REPLOG utility.

THE AGENT COMPONENT

The LMS Agent component must satisfy the requests for network management services (NMS), using the tools supplied by the Server component(s) of the system(s) monitored (see the "NMS" section in the "Networks" chapter).

OTHER MAINTENANCE TOOLS

To help the software support specialist in problem solving, a number of maintenance commands, a system debugger, and some further utilities are provided.

MAINTENANCE COMMANDS

CSIZE — Displays the amount of memory occupied by an executable program in each segment in which it resides, and may also display the message field.

DISKCHECK — Checks the data tables that record the contents of a disk volume and outputs information on file descriptors (PDDs), file extents, and disk block usage.

DISKEDIT — Allows changes to be made to the bit map, the extent tables, the PDD table, the root directory, and volume descriptors on the disk.

HEXED — Allows a file containing hexadecimal characters to be displayed and modified.

LISTCON — Gives access to the system configuration file $CON.

LISTMSG — Lists the header messages of the l-modules in a specified directory or volume (for example, the system volume) to obtain information on their contents and the dates when they were linked.

NETNOSE — Provides information on resource availability and machine status across a local area network.

NOSE — Provides detailed information about the current state of the operating system and its components.

PERFPROG — Appraises system performance.

TREECHECK — Checks whether the index (B*-tree) of a keyed or packed positional file is still working, and checks the level at which the file can be accessed.

TREEMEND — Rebuilds the index (B*-tree) of a keyed or packed positional file that has been damaged during a system crash.

VOLGC — Restores any damaged data structures on the volume.

SYSTEM DEBUGGER

A system debugger is available (in ROM and RAM versions), which provides information on the contents of specific memory blocks, registers, and so on.

TOTAL MEMORY DUMP

If the operating system crashes, the user can start a total memory dump, carrying out a series of operations described in the manual System Software Maintenance, User Guide.

He can then use the TMDUMP utility to interpret the data in the memory image that will have been saved on the floppy disk.

THE READALTM UTILITY

To complete the functions described above, the READALTM utility can be used to display the contents of family segments and to modify single locations in memory.

For further details on the above commands and utilities, and on how to maintain the system software, see the manual System Software Maintenance, User Guide.

For a description of the NETNOSE utility, see the manual Distributed System in Local Area Network (LAN), User Guide.

RUN-TIME DIAGNOSTICS

By using the run-time diagnostics, all the functions of some of the hardware modules that make up a work station may be interrogated to check that they are fully and correctly operational.

Run-time diagnostics comprise a set of programs that can be executed simultaneously with normal system activities; that is, without interrupting user activities on other work stations.

This type of diagnosis may be applied to video terminals, badge readers, and printers.

14. SYSTEM PROGRAMMING TOOLS

This chapter introduces the main system programming tools provided under MOS.

INTRODUCTION

PASCAL+ is used as the main system programming language on MOS systems. PASCAL+ programs can interface directly with Operating System components by means of procedure and function calls, without the need to use additional run-time system routines. The system programmer can use these interfaces directly to construct system software.

PASCAL+ PRINCIPAL CHARACTERISTICS

PASCAL+ is an extension of the ISO standard PASCAL programming language. It was developed as a tool for system programming.

PASCAL+ introduces the concepts of modules and monitors into standard PASCAL.

A **module** is a resource library with the same structure as a main program unit, but without the instruction part. It may contain variables, constants, types, and routines that are defined within the module. Objects defined within a module must be explicitly exported by the module in which they are defined, and explicitly imported by all the program components in which they are to be used. This mechanism ensures that all interfaces are well defined, and can be controlled by the compiler.

Monitors are encapsulation mechanisms for the control of shared data in concurrent programs. A monitor is the same as a module, with the exception of a scheduling rule that determines the order in which processes can access the monitor.

The main facilities offered by monitors are:

- mutual exclusion of concurrent processes
- synchronisation of cooperating processes.

This is achieved by means of variables of the predefined type condition , used as a mechanism for signalling the availability of the information to be communicated.

Variables of the **condition** type are used to schedule the access of processes to a monitor, so that only one process has access to a particular monitor at any one time.

Access to the monitor is made through a call to a monitor access procedure. Within this procedure, a variable of the predefined type CONDITION is used to delay the access of the calling process if the monitor is already in use. When the monitor is freed by its calling process, the next process in the queue (if any) is able to use it.

Variables of the CONDITION type may be used together to permit scheduling regimes of arbitrary complexity.

The remaining parts of PASCAL+, with some minor exceptions such as conformant array and strings of non-alphanumeric characters, are exactly as defined in the ISO Standard. PASCAL+ thus contains the standard PASCAL language as a subset.

Standard PASCAL programs may be written in PASCAL+. This is achieved by omitting from the program text all the features that have been added to the standard language.

PASCAL+ program units may be separately compiled: that is, modules and monitors may be compiled separately, and stored in a library until link time. Alternatively, PASCAL+ source text may be included in the text of other PASCAL+ programs by means of an include directive.

EDITING PASCAL+ PROGRAMS

PASCAL+ source programs are created on L1 MOS Systems by using the standard editor provided with the system, and then submitted to the PASCAL+ compiler.

COMPILING

The PASCAL+ compiler is able to compile program components separately. It is also able to include source text from files other than the one currently being compiled. That is, source text may be inserted into the text of the currently compiling program module from other files of source text.

The PASCAL+ compiler permits the user to insert compiler directive lines into the source text of a program. The compiler directive lines concern such matters as the inclusion of debugging code into the object module produced by the compiler, the production of a listing file, and the redirection of the listing file to a user-selected file.

It is also possible to control the allocation of global variables on the stack: this is done when a PASCAL+ program is composed of a number of separately compiled modules or monitors. The code produced by the PASCAL+ compiler is fully re-entrant.

LINKING

After a compilation, PASCAL+ programs must be linked to a run-time library, and to separately compiled units of the program (if any)..

On L1 MOS systems, linking is performed by either one of two linkers, **ZLOC** and **OLINK.** The **ZPC** compiler normally invokes the **ZLOC** linker automatically; therefore, if **OLINK** is required, the compiler must be invoked with the -c option, to inhibit the automatic linking phase. Both linkers accept any number of object files and produce as output a MOS 1-module or program directory.

DEBUGGING

It is possible to debug PASCAL+ programs on the L1 MOS Systems by using a symbolic debugger.

All PASCAL+ compilation units to be debugged must be compiled with the -d option (on the command line that invokes the compiler). Two files are then produced by the compiler, with information about PASCAL+ code and declarations. These two files must be notified to the symbolic PASCAL+ debugger, **SPD**, at run time.

The programmer is provided with complete control over PASCAL+ statements, procedures, and variables by a set of very simple debugging commands, which enable him to monitor the flow of program execution.

I. INDEX

Code 4002130 G (5)
Printed In Italy