COBOL

Language
User's Guide
Part 1 of 3

A6L0113

Lisa

COBOL

COBOL User's Guide for the Lisa™

029-0306-A

Licensing Requirements for Software Developers

Apple has a low-cost licensing program, which permits developers of software for the Lisa to incorporate Apple-developed libraries and object code files into their products. Both in-house and external distribution require a license. Before distributing any products that incorporate Apple software, please contact Software Licensing at the address below for both licensing and technical information.

20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010

Customer Satisfaction

If you discover physical defects in the manuals distributed with a Lisa product or in the media on which a software product is distributed, Apple will replace the documentation or media at no charge to you during the 90-day period after you purchased the product.

Product Revisions

Unless you have purchased the product update service available through your authorized Lisa dealer, Apple cannot guarantee that you will receive notice of a revision to the software described in this manual, even if you have returned a registration card received with the product. You should check periodically with your authorized Lisa dealer.

Limitation on Warranties and Liability

All implied warranties concerning this manual and media, including implied warranties of merchantability and fitness for a particular purpose, are limited in duration to ninety (90) days from the date of original retail purchase of this product.

Even though Apple has tested the software described in this manual and reviewed its contents, neither Apple nor its software suppliers make any warranty or representation, either express or implied, with respect to this manual or to the software described in this manual, their quality, performance, merchantability, or fitness for any particular purpose. As a result, this software and manual are sold "as is," and you the purchaser are assuming the entire risk as to their quality and performance.

In no event will Apple or its software suppliers be liable for direct, indirect, special, incidental, or consequential damages resulting from any defect in the software or manual, even if they have been advised of the possibility of such damages. In particular, they shall have no liability for any programs or data stored in or used with Apple products, including the costs of recovering or reproducing these programs or data.

The warranty and remedies set forth above are exclusive and in lieu of all others, oral or written, express or implied. No Apple dealer, agent or employee is authorized to make any modification, extension or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights that vary from state to state.

License and Copyright

© 1983 by Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010

Apple, Lisa, and the Apple logo are trademarks of Apple Computer, Inc.

Simultaneously published in the USA and Canada.

Reorder Apple Product # A6D0104 (Complete COBOL package)
A6L0113 (Manuals only)

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the CODASYL Programming Language Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection herewith.

The authors and copyright holders of the copyrighted material used herein:

> FLOW-MATIC (Trademark for Sperry Rand Corporation) Programming for the Univac(R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F28-8013, copyrighted 1959 by IBM; FACT, DS127A5260-2760, copyrighted 1960 by Minneapolis-Honeywell.

have specifically authorized the use of this material, in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

Contents

Tables

Figures

Preface

The *COBOL User's Guide for the Lisa* describes operating procedures for the Lisa resident releases of the LEVEL II COBOL compiler and run-time libraries. The compiler converts LEVEL II COBOL source code into intermediate code, which is interpreted at run time. The steps needed to compile and execute a program are described, including all necessary linkage, relocation, and run-time requirements.

Note that within this manual the product name LEVEL II COBOL Version 2 is occasionally abbreviated to L/II COBOL, and that COBOL as defined in the ANSI standard X3.23 1974 is referred to as ANSI COBOL.

Audience

This manual is intended for personnel already familiar with COBOL.

Manual Organization

Chapters 1 through 5 of this manual describe compiler features and general procedures for loading and executing programs.

The appendixes provide summarized information for reference purposes.

This manual contains the following chapters and appendixes:

Chapter 1. Introduction, gives a general description of the LEVEL II COBOL system, its input and output files, and the run-time libraries provided with the compiler, plus the step-by-step outline of compilation and execution of sample interactive programs.

Chapter 2. Compiler Controls, describes compiler commands, directives, and listing formats.

Chapter 3. Run-Time System Controls, gives general instructions for running programs, operating the console, and CRT screen handling.

Chapter 4. CRT Screen Handling, describes in detail the extended ACCEPT and DISPLAY facilities provided in LEVEL II COBOL for easy manipulation of data with the CRT.

Chapter 5. Program Design Considerations, describes the facilities available to overlay programs and invoke other COBOL programs.

Appendix A. Summary of Compiler and Run-Time Directives, summarizes the compiler directives available in the LEVEL II COBOL compiler.

Appendix B. Compile-Time Errors, lists all errors that can be signaled during program compilation.

Appendix C. Run-Time Errors, lists all errors that can be signaled during program execution.

Appendix D. File Formats, describes device and file naming conventions and formats used by LEVEL II COBOL.

Appendix E. Useful Facts and Figures, briefly lists data that might be useful in designing or debugging a LEVEL II COBOL program.

Appendix F. COBOL Workshop Files, lists the names of the files contained on the COBOL language disks.

Related Publications

For details of the LEVEL II COBOL Language, refer to the *COBOL Reference Manual for the Lisa.*

For details of the Lisa Operating System, Messages, and File Structures refer to the *Workshop User's Guide for the Lisa.*

Notation in this Manual

Throughout this manual the following notation is used to describe the format of data input or output

1. All words printed in lowercase letters are generic terms representing names devised by the programmer.
2. When material is enclosed in square brackets [ ], it is an indication that the material is an option which can be included or omitted as required.
3. The symbol << after a CRT entry or command format in this manual indicates that the [RETURN] key must be pressed to enter the command.
4. All numbers are in decimal unless otherwise stated.
5. In the sample screen "conversations" using the CRT shown in this manual, displays are shown as they occur with the user response underlined. Because underlining is a convention used to differentiate user response from system response, the user does not include it as part of the response.

Headings are presented in the following order of importance:

Chapter Title

n.n Order One Heading
n.n.n Order Two Heading
n.n.n.n Order Three Heading
Order Four Heading

} Text beneath

Chapter 1
Introduction

Introduction

1.1 General Description

COBOL (Common Business Oriented Language) is the most widely and extensively used language for the programming of commercial and administrative data processing.

LEVEL II COBOL is a compact, interactive, and standard COBOL language system designed for use on microprocessor-based computers and intelligent terminals under control of the Lisa Operating System.

The LEVEL II COBOL compilation system converts LEVEL II COBOL source code into an intermediate code, which is then interpreted by a Run-Time System (RTS).

LEVEL II COBOL programs can be created using a standard Lisa text editor to create the LEVEL II COBOL source files, from which the compiler compiles the source programs. A listing of the LEVEL II COBOL program and any error messages is provided by the compiler during compilation.

The user should be familiar with the Lisa Operating System (see *Workshop User's Guide for the Lisa*) prior to beginning this manual.

1.2 Getting Started with LEVEL II COBOL

1.2.1 Issue Disk

The issue disk provides each user with the software that makes up the COBOL development system described above. The contents of this software package are listed in Appendix F.

1.2.2 The Compiler

The LEVEL II COBOL compiler contains several overlays and loads each overlay file from the diskette. The root segment is contained in COBOL.INT, and the overlays are contained in the other COBOL files.

1.2.3 The Run-Time System

The Run-Time System (RTS) executes the intermediate code generated by the compiler. In addition to standard ANSI COBOL statements, LEVEL II COBOL contains many extensions for use with interactive programs on the Lisa.

1.2.4 The Demonstration Programs

PI.TEXT, STOCK1.TEXT, and STOCK2.TEXT are simple demonstration programs, supplied in source form, which show many of the facilities present in LEVEL II COBOL, and which can be used to become familiar with the system.

1.2.5 The COBOL Command Line

The COBOL command line appears on the top line of the system starting screen and provides a choice of actions. Select one by entering a single letter command; for example, C for Compile, R for Run, S for Set Switches.

1.2.6 First Steps

1.2.6.1 Initialization

To obtain a working LEVEL II COBOL system, follow the installation instructions provided in Chapter 1 of the *Workshop User's Guide for the Lisa*.

1.2.6.2 Device Management

For compilation, the compiler.INT files must reside on the boot volume. By default the intermediate code is output to the disk containing the source at compilation. The RTS (COBOL.OBJ) can reside on any of the default working directories the user chooses. The most efficient disk allocation for this system is the user's responsibility.

1.2.6.3 Compilation

Compile all the demonstration programs, which are the source files with the extension .TEXT. If the user knows that a source file to be compiled ends in .TEXT, this extension can be omitted when entering the source file name.

Example:

```
LEVEL II COBOL:  Compile, Run, Set Switches, Printer, Quit : C
COBOL Source file [.TEXT] - STOCK1 <<
Compiler directive -  <<
*LEVEL II COBOL Vv.r (C) 1982 Micro Focus, Ltd 1983 Apple Computer,Inc.
*Compiling STOCK1.TEXT
*ERRORS=00000 DATA=nnnn CODE=nnnnn DICT=nnnnn:nnnnn/nnnnnGSAFLAGS=OFF

    LEVEL II COBOL:  Compile, Run, Set Switches, Printer, Quit :
```

After compilation, a directory listing of the disk shows that two new files exist: STOCK1.LST.TEXT, which is the list file, and STOCK1.INT, which is the file containing the intermediate code. Follow the same procedure to compile STOCK2 and PI.

Note that STOCK2 has an error in it which is present to show error formats. This error is for demonstration purposes only, and does not affect the running of the program.

The message produced by the error is:

```
nnnnnn    MOVE GET-INPUT TO TF-DATE.
**103************************                    **
**   Wrong data type or data name not declared.        **
```

1.2.6.4 Running the Demonstration Programs

By compiling and running the demonstration programs, the user has checked the disk and mastered the fundamentals of LEVEL II COBOL facilities. If the user knows that a program to be run ends in .INT, this extension can be omitted when entering the COBOL program name.

Calculation of PI:

```
LEVEL II COBOL:  Compile, Run, Set Switches, Printer, Quit : R
Run what COBOL program? - [.INT] PI <<
Run time directive : <<
```

The screen clears, followed by:

```
CALCULATION OF PI

NEXT TERM IS 0.000000000000

PI IS  3.141592653589

LEVEL II COBOL:  Compile, Run, Set Switches, Printer, Quit:
```

During the execution of PI the next term changes as the iteration progresses.

Stock Control Program One (Cursor Control):

```
LEVEL II COBOL:  Compile, Run, Set Switches, Printer, Quit : R
Run what COBOL program?  - [.INT] STOCK1 <<
Run time directive : <<
```

The screen clears, followed by:

```
STOCK CODE      <    >
DESCRIPTION     <                         >
UNIT SIZE       <    >
```

STOCK1 is a skeleton stock data entry program in which stock records are created on a stock file in stock code order. This program provides an opportunity to use the cursor control functions. The user has the ability to:

- tab the cursor forwards and backwards from one data input field to the next.
- move the cursor backwards and forwards nondestructively one character position at a time in data input fields.
- place the cursor HOME to the first character position in the first data input field.

In addition, numeric validation, which permits only numeric characters to be entered on numeric fields, is available. Left zero fill on numeric fields is automatic. See Cursor Control Facilities in Section 4.2.1 for cursor control keys on the Lisa keyboard.

Running STOCK1 also creates an indexed sequential file on disk called STOCK.IT together with its index called STOCK.IDX.

To create a record, key data into the unprotected areas defined by < >. When the record is complete, press the [RETURN] key and the record is written to disk. If the record has been correctly entered, the unprotected areas are space filled, ready for the next record to be entered. If the record remains displayed, the record was incorrectly keyed and should be entered again.

To terminate the run, enter spaces into the STOCK CODE field and press the [RETURN] key.

The result is:

```
END OF PROGRAM
```

Stock Control Program Two (Data Input) is run the same way.

After the user responds to the Run command, the screen clears, followed by:

```
                GOODS INWARD

STOCK CODE      <    >
ORDER NO        <      >
DELIVERY DATE MM/DD/YY
NO OF UNITS     <    >
```

STOCK2 is a skeleton stock data input program by which the stock records created by STOCK1 can be accessed.

The cursor control features are the same as in STOCK1.INT. Note, however, that the DELIVERY DATE has a different method of prompting than was previously used.

Stop the same way as for STOCK1.

1.3 Program Development Cycle

The cycle for developing and running LEVEL II COBOL application programs is shown in Figure 1-1.

Preparation: The source programs are created on diskette with the user's own existing editor program.

Compilation: The following procedure ...

Compile, Run, Set Switches, Printer, Quit: C
COBOL Source file [-TEXT] - MYPROG <<
Compiler directive - <<

... loads the single pass compiler to convert a source program (MYPROG.TEXT in this example) into an object form known as Intermediate Code (MYPROG.INT). The user can specify the file on which the listing is to appear. If this is a disk file, it can be edited to correct errors and used as input for the next run of the compiler.

Running: The following procedure ...

Compile, Run, Set Switches, Printer, Quit: R
Run what COBOL program? - [.INT] - MYPROG <<
Run time directive - <<

... loads the Intermediate Code, which is then run.

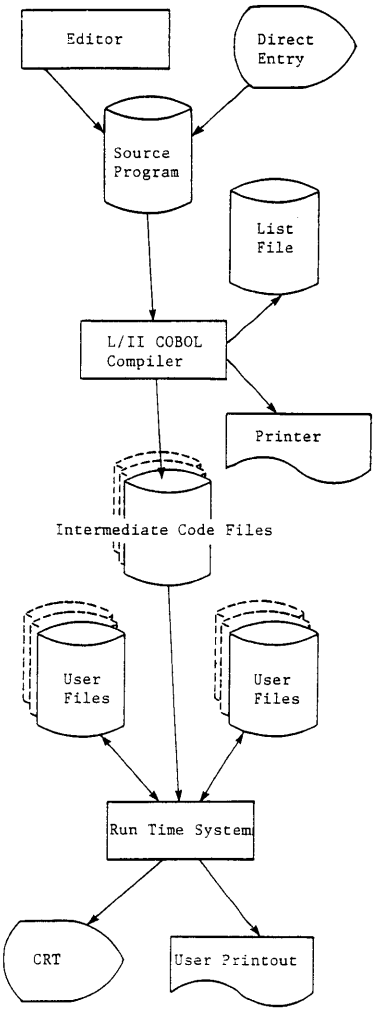


Figure 1.1
Program Development Cycle

1.3.1 Program Preparation Considerations

The LEVEL II COBOL compiler accepts source input from a standard source file, specified on the compiler command line, as produced by any standard Lisa editor or compatible proprietary editor software.

The LEVEL II COBOL program format is the same as standard COBOL and is detailed in the *COBOL Reference Manual for the Lisa*.

NOTE

1. Each line of source code, including the last line, must be terminated by pressing the [RETURN] key.
2. The compiler rejects most nonalphanumeric characters within the input file; for example, the [TAB] character, unless embedded in literal strings.

1.3.2 Program Design Considerations

LEVEL II COBOL provides the full COBOL facilities for overlaying in memory and for dynamically invoking programs or subroutines, whether written in COBOL or assembly languages, as specified in the COBOL Segmentation and Interprogram Communication modules. Chapter 5 contains more information on the use of these features.

Chapter 2
Compiler Controls

Compiler Controls

2.1 Command Line Syntax

The COBOL command line is where the user

1. invokes the COBOL compiler,
2. specifies the name of the source file to be compiled, and
3. enters the directives that modify the way in which the compiler processes the source file.

The command line can be continued by using the ampersand (&) character.

The format of the command line is:

```
LEVEL II COBOL:  Compile, Run, Set Switches, Printer, Quit : C
COBOL Source file [.TEXT] -   filename <<
Compiler directive  -  directive <<
        .    .
        .    .
Compiler directive - <<
```

where:

filename — is the name of the program that contains the LEVEL II COBOL source statements to be compiled. The default extension is .TEXT.

directives — is a sequence of LEVEL II COBOL directive statements. Directives are supplied one at a time by the user until a blank line is supplied by pressing the [RETURN] key.

The general form of directives is:

[NO] keyword [argument]

NO — Most directives can be "switched off" by use of the word NO before the keyword. NO can adjoin the keyword or be separated from it by one or more spaces. NO is permitted where specified in the list of directives below.

keyword — One of the directives listed below.

argument Where applicable, argument is a qualifier to the keyword. The argument must appear in one of two forms, and can adjoin the keyword or be separated from it by one or more spaces.

"argument" Where quotes are used the argument can contain spaces.

(argument) Where parentheses are used no spaces are permitted.

NOTE

The commands are processed in order of entry, and a directive or its negative can appear more than once; the setting of the directive used by the compiler is the setting encountered last. This rule does not apply where a directive is used but is excluded by the use of another directive. See Table 2-1.

2.2 Compiler Directives

[NO] ALTER NO ALTER prohibits the use of ALTER statements within the program being compiled, allowing the compiler to operate more efficiently.

The default is ALTER.

[NO] BRIEF Error numbers only to be produced on the listing and console; that is, the text of error messages is suppressed.

By default this directive is off, meaning that error messages are printed, unless no error message file can be found.

[NO] COMP Causes the compiler to generate much more compact and efficient code for certain statements involving PIC 9(2) COMP and PIC 9(4) COMP data items. See Chapter 5 for full details. The reason for this directive is that the efficient code leads to nonstandard behavior in cases of numeric overflow; the compiler cannot allow this to happen unless the user specifies this directive, meaning that either the user knows the statements do not lead to numeric overflow (in which case the semantics of the program remain strictly in accord with the ANSI standard while at the same time giving the advantage of the extra efficiency), or alternatively the user means to take advantage of the defined but nonstandard behavior that occurs on overflow.

By default this directive is off.

[NO] COPYLIST Causes the contents of any files named in COPY statements to be listed.

By default this directive is off.

Whatever the state of this directive, the name of any copy file open at the time a page heading is output is listed as part of the heading.

COPYLIST causes COPYLIST to be set in the IDENTIFICATION DIVISION and in Segment 53 but not otherwise.

NO COPYLIST causes COPYLIST to be set in Segment 53 only.

CRTWIDTH "integer"

Specifies the width of the user screen in characters. This directive is used in Format 1 (standard ANSI) DISPLAY statements to enable the user to plan the separation points in displaying data items too long to fit on one physical CRT line. The "integer" must be between 40 and 255.

By default this directive is set to 128.

DATE "string"

Causes the "string" to be used in place of the comment entry in the DATE-COMPILED paragraph, if present.

If the directive is omitted the comment entry, if present, is used.

[NO] ECHO Causes error lines and flags to be echoed to the console. When an error occurs, the source line producing it, the error number, and (unless BRIEF is set) an explanatory message are printed on the console.

By default this directive is on.

[NO] ERRLIST Causes the listing to be restricted to those COBOL lines containing syntax errors or flags, together with associated error messages.

By default this directive is off.

[NO] FLAG " LOW | L-I | H-I | HIGH | L/II | IBM "

Causes the output of GSA compiler certification flags during compilation for all features higher than the specified level.

LOW	GSA Low-level
L-I	GSA Low-Intermediate-level
H-I	GSA High-Intermediate-level
HIGH	GSA High-level
L/II	LEVEL II COBOL extensions to ANSI COBOL standard X3.23 1974. See the *COBOL Reference Manual for the Lisa.*
IBM	IBM-compatible nonstandard COBOL. See Appendix J, *COBOL Reference Manual for the Lisa.*

By default this directive is off.

[NO] FORM "integer"

Specifies the number of lines per page of the listing. The "integer" must be at least 3.

By default 60 lines per page are printed.

One formfeed character is always produced at the head of the listing file. If NO FORM is used, no further formfeed characters and no page headings are produced in the body of the listing.

If the listing is directed to the console, by use of the LIST directive, then the first formfeed character is replaced by a blank line.

[NO] INT "filename"

Specifies the file to be used to hold the intermediate code output by the compiler. If the file specified exists, it is overwritten.

NO INT suppresses the production of an intermediate code file; that is, the compiler is used for syntax checking only.

By default the compiler adds .INT to the source file name, replacing any existing file name extension.

[NO] {LIST / PRINT} "destination"

Specifies the destination of the listing file. If an existing file is specified, it is overwritten. The destination can be a printing device; for example, the printer or the console.

NO {LIST / PRINT} suppresses production of a listing.

If "destination" is "-console", the listing is directed to the console. If "destination" is "-printer", the listing is directed to the printer.

If no directive is specified, the compiler forms a file name by adding .LST.TEXT to the source file name. If a directive is specified, but no file name given, then the console is used.

[NO] QUAL — NO QUAL prohibits qualified data names or procedure names from the program being compiled, allowing the compiler to operate more effectively.

The default is QUAL.

[NO] REF — Causes four-digit location addresses to be included on the righthand side of the listing file. Note that a listing with location addresses can be required in order to identify the locations reported in RTS error messages.

By default this directive is off.

[NO] RESEQ — Causes the compiler to generate COBOL line sequence numbers in increments of 10, starting at 10.

By default this directive is off.

2.2.1 Excluded Combinations of Directives

Using certain directives implies that certain other directives are ignored, even if specified. Table 2-1 shows the combinations that are not permitted.

Table 2-1
Excluded Combinations of Directives

Directive	Excluded Directives
ERRLIST	COPYLIST [NO] REF RESEQ
[NO] LIST	COPYLIST ERRLIST [NO] FORM LIST PRINT [NO] REF RESEQ

2.3 Reference Table of Directives

Table 2-2
Reference Table of Directives

Directive	Use	Default
[NO] ALTER	Allow ALTER statements	ON
[NO] BRIEF	Suppress error messages	OFF
[NO] COMP	Use computational subset	OFF
[NO] COPYLIST	List COPY files	OFF
CRTWIDTH "n"	Set width of CRT to "n"	ON n = 128
DATE "string"	As DATE below but "string" set to spaces Use "string" for comment entry in DATE-COMPILED paragraph	ON
[NO] ECHO	Echo errors to console	ON
[NO] ERRLIST	List only errors and flags	OFF
[NO] FLAG "[LOW / L-I / H-I / HIGH / L/II / IBM]"	Flag code higher than level indicated	OFF
[NO] FORM "n"	Suppress headers and form-feeds Set length of page = "n" lines	ON n=60
[NO] INT "filename"	Specify intermediate code filename	ON = source filename

Directive	Use	Default
[NO] {LIST / PRINT} "filename"	Specify listing requirements	If no directive: ON; that is filename = source filename. If directive but no filename: filename = -console
[NO] QUAL	Allow qualified data-names and procedure-names	ON
[NO] REF	Insert addresses on listing	OFF
[NO] RESEQ	Resequence source file	OFF

2.4 Summary Information on CRT

After the user completes the command line, the compiler replies with:

**LEVEL II COBOL Vv.r (C) 1983 Apple Computer,Inc. 1982 Micro Focus, Ltd

where v is the version number and r is the release number.

The compiler then acknowledges each directive on a separate line, and either accepts or rejects it. If the command line is continued using the ampersand (&) character, each line of directives is processed before the compiler allows the next line to be entered. After the compiler acknowledges all the directives, it opens its files and starts to compile. At this point it displays the message:

```
* compiling filename.Text
```

If any file fails to open correctly, the compiler displays:

```
Open fail: filename.Text
```

Compilation is terminated, and control returns to the Operating System.

When the compilation is complete, the compiler displays the message:

**ERRORS=nnnnnDATA=nnnnnCODE=nnnnnDICT=mmmmm:nnnnn/pppppGSAFLAGS=nnnnn

where:

ERRORS	denotes the number of errors found.
DATA	denotes the size of RAM required; that is, data area of the generated program.
CODE	denotes the size of ROM required; that is, code area of the generated program.
DICT	mmmmm denotes the number of bytes used in the data dictionary. nnnnn denotes the number of bytes remaining in the data dictionary ppppp denotes the total number of bytes in the data dictionary
GSA FLAGS	denotes the number of compiler validation flags encountered or 'OFF' if the directive NOFLAG was entered or assumed.

2.5 Listing Formats

The general format of the list file is:

```
*LEVEL II COBOL Vv.r (C) 1983 Apple Computer, Inc. 1982 Micro Focus, Ltd
* Accepted
* Rejected - optional directive as entered in compile command line
* Compiling filename
* LEVEL II COBOL Vv.r          filename          Page: nnnn
*
statement 1                                      HHHH
        .                                          .
        .                                          .
        .                                          .
statement n                                      HHHH
* LEVEL II COBOL Vv.r revision n                URN AA/0000/AA
* Compiler (C) 1983 Apple Computer, Inc. 1982 Micro Focus, Ltd.
*
*ERRORS=nnnnn DATA=nnnnn CODE=nnnnnDICT=mmmmm:nnnnn/pppppGSAFLAGS=OFF
```

The first two lines of title information are repeated for each page. The final line is the same as on the CRT display. The value denoted by HHHH is a hexadecimal value denoting the address of each data name or procedure statement, and is generated if the REF directive is specified in the command line. Addresses of data names are relative to the start of the data area, while addresses of procedure statements are relative to the start of the code area. An overhead is at the start of the data area, and a few bytes of initialization code are at the start of the procedure area for each SELECT statement defined in the ENVIRONMENT DIVISION.

A syntax error is marked in the listing by an error line with the following format:

```
nnnnnn          illegal statement
** nnn *** ...          ... ***                **
```

where:

nnnnnn is the sequence number of the erroneous line.

nnn denotes the error number.

The asterisks following the error number indicate the character position of the error in the preceding erroneous source line, while the asterisks at the end of the line simply highlight the error line.

NOTE

The demonstration program STOCK2, compiled as described under Compilation in Chapter 1, contains a sample error line.

A flag is marked in the listing by a flagging line with the following format:

```
nnnnnn                  flagged feature
** level ---            . . . ---                -------
```

where:

nnnnnn is the sequence number of the flagged line.

'level' represents the level at which the feature is flagged using the same acronyms as can be entered in the command line, when setting the lowest required flagging level:

LOW	Low level
L-I	Low-Intermediate level
H-I	High-Intermediate level
HIGH	High level
L/II	LEVEL II COBOL extensions
IBM	IBM-compatible extensions

The flagged feature is pinpointed at the end of the line of characters beneath the flagged line. The dashes at the end of the line simply highlight the flagging line.

NOTE

A program in which flags are indicated can still be run. However, errors should always be corrected, and the program recompiled, before the object program is run.

Chapter 3
Run-Time System Controls

Run-Time System Controls

3.1 Run Time Directives

3.1.1 Command Line Syntax

The COBOL command line syntax for running a LEVEL II COBOL object program is:

```
LEVEL II COBOL:  Compile, Run, Set Switches, Printer, Quit : R
Run what COBOL program? - [.INT] :  filename.INT<<
Run time directive - directives<<
```

filename is the name of the intermediate code file. An example of the whole RUN command is given later in this chapter.

The filename is of the form

```
name.INT
```

Note that the search is of the volume specified in the filename. If no volume is specified the prefixed ones are searched. If filename is not found, the user is prompted for the correct filename and, simply presses [RETURN] to exit to the COBOL command line.

3.1.1.1 Switch Parameter

LEVEL II COBOL includes the facility of controlling events in a program at run time depending on whether or not programmable switches are set by the user. See the description of the SPECIAL-NAMES paragraph in Section 3.4.1.2 of the *COBOL Reference Manual for the Lisa.* The user sets these switches at run time with the Switch option in the COBOL command line. When switches have been set at run time, they remain set when COBOL CALLed modules are processed.

Example:

```
LEVEL II COBOL: Compile, Run, Set Switches, Printer, Quit : S

Current Switch settings

   SW-0  SW-1  SW-2  SW-3  SW-4  SW-5  SW-6  SW-7
   OFF   OFF   OFF   OFF   OFF   OFF   OFF   OFF
```

```
Do you wish to change the settings?: y

    SWITCH SW:0 = OFF .FLIP? y
    SWITCH SW:1 = OFF .FLIP? n
    SWITCH SW:2 = OFF .FLIP? n
    SWITCH SW:3 = OFF .FLIP? y
    SWITCH SW:4 = OFF .FLIP? y
    SWITCH SW:5 = OFF .FLIP? n
    SWITCH SW:6 = OFF .FLIP? n
    SWITCH SW:7 = OFF .FLIP? n

Updated Switch settings

    SW-0  SW-1  SW-2  SW-3  SW-4  SW-5  SW-6  SW-7
    ON    OFF   OFF   ON    ON    OFF   OFF   OFF

Current ANSI Debug IS switched OFF

Do you wish to change the setting?: y

    Updated ANSI Debug IS switched ON

LEVEL II COBOL: Compile, Run, Set Switches, Printer, Quit :
```

The switches remain in this state until remodified or until the user exits the COBOL environment.

3.1.1.2 Standard ANSI COBOL Debug Switch Parameter

Users can also include a parameter to invoke the standard ANSI COBOL Debug module. See Chapter 11 of the *COBOL Reference Manual for the Lisa* for a description of the Debug facilities.

To include the standard ANSI Debug facility a run-time switch is required. The format is the same as for a normal switch parameter (see Switch Parameter above).

3.1.1.3 Program Parameters

Program parameters are any parameters required by the program. They can be read in on the console file device "-console". There can be as many parameters as the programmer requires, text or numbers in any format, limited to 80 characters on the command line.

Two methods access the program parameters:

READ
ACCEPT

READ

"-console" can be declared as a sequential or line sequential file (line sequential is preferable as fixed-length records are not then expected). This file can then be accessed by a READ; READ returns the command

line program parameters until the command line is exhausted. Subsequent READs expect console input. In the absence of command line parameters, the first READ returns SPACES.

ACCEPT

The first ACCEPT FROM CONSOLE statement in a program returns the program parameters from the command line. The CRTWIDTH directive affects the behavior of the ACCEPT statement: ACCEPT is compiled as a sequence of reads, each of CRTWIDTH characters, sufficient to fill the data item specified.

Subsequent ACCEPT statements expect console input.

Chapter 4
CRT Screen Handling

CRT Screen Handling

4.1 Introduction

COBOL is traditionally a batch processing language, allowing for one line of data at a time to be read into memory from the console, and for one line to be displayed at a time. LEVEL II COBOL extends the language to make it fully interactive; that is, whole screens of data can be displayed or entered into memory using just one statement.

The ACCEPT and DISPLAY statements, used in COBOL for one line input and output, are extended in LEVEL II COBOL to provide further facilities for interaction with the user.

The remainder of this chapter describes the use of these facilities; for a detailed specification refer to The ACCEPT Statement, The DISPLAY Statement, and The SPECIAL-NAMES Paragraph in Section 3.4.1.3 in the *COBOL Reference Manual for the Lisa*.

4.2 Using the Extended ACCEPT and DISPLAY Statements

The extended formats of ACCEPT and DISPLAY are:

```
ACCEPT  data-name-1 [ [ AT { data-name-2 } ]  FROM CRT ]
                           { literal-1   }
                    [  AT { data-name-2 }                ]
                          { literal-1   }

DISPLAY { data-name-1 } [ [ AT { data-name-2 } ]  UPON CRT ]
        { literal-3   }         { literal-4   }
                        [  AT { data-name-2 }              ]
                              { literal-4   }
```

Using these statements is described in the remainder of this chapter.

If these formats are to be used extensively, then the user might want to make them the default formats of ACCEPT and DISPLAY in the LEVEL II COBOL program, and thus not have to specify AT ... FROM CRT or UPON CRT every time.

In order to do this, the statement

CONSOLE IS CRT.

should be included in the SPECIAL-NAMES Paragraph of the ENVIRONMENT DIVISION of the LEVEL II COBOL program. The following sections assume that this has been done.

Example:

```
.
.
.
SPECIAL-NAMES.
CONSOLE IS CRT.
.
.
.
```

4.2.1 Operator Cursor Control Facilities

Interaction implies action on the part of the user during the execution of a program. The user has control over the cursor while data are being entered in response to an ACCEPT statement in the program. The cursor is manipulated on the CRT screen by the cursor control functions on the console keyboard of your CRT device as shown in Table 4-1.

Table 4-1
CRT Cursor Control Keys

Cursor Movement	Key Function
Tab forward a field	↓
Tab backward a field	↑
Forward Space	→
Backward Space	←
Column Tab	TAB
Left Zero Fill [1]	.
Return	RETURN

1 - The "." for left zero fill is a "," when

DECIMAL-POINT IS COMMA.

is specified in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

NOTE

Although the functions defined above are available on most keyboards, the actual keys required to generate them can vary. Check with the documentation supplied with the console keyboard.

4.3 Displaying Data on the Screen

The first step in making the LEVEL II COBOL program interactive is to decide what messages and prompts are to be displayed on the screen to guide the user, and what action the user is to take at each point. This section describes the display facilities.

NOTE

As most terminals scroll upwards as a result of a character appearing in the final character position (that is, bottom right of the screen), this character position cannot be used as part of a DISPLAY.

4.3.1 Clearing the Screen

Unless deliberately displaying something upon a screen, which has already been displayed and the results are known, the user should clear the screen before any display. The statement

```
        DISPLAY SPACE.
or      DISPLAY SPACES.
```

causes the entire screen to be cleared.

4.3.2 Displaying Single Items

Single text strings; for example, single prompts or messages, can be displayed easily by using the AT clause to specify the coordinates of the start of the display item on the screen:

```
                                  {integer    }
DISPLAY data-item-1 AT            {data-item-2} .
```

where data-item-2 is PIC 9999. The most significant two digits specify a line number in the range 01 to the maximum number of lines on the screen, and the least significant two digits specify a column number in the range 01 to the maximum number of characters per line on the screen. Both numbers are in decimal.

Data-item-1 is the text to be displayed.

Example:

```
ENVIRONMENT DIVISION.
SPECIAL-NAMES.
CONSOLE IS CRT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  DISPLAY-ITEM-1  PIC X(33)
        VALUE "SELECT ONE OF THE FOLLOWING ITEMS".
        .
        .
        .
PROCEDURE DIVISION.
START-OF-PROGRAM.
        DISPLAY SPACES.
        DISPLAY DISPLAY-ITEM-1 AT 0507.
```

causes the message SELECT ONE OF THE FOLLOWING ITEMS to be displayed on line 5 of the screen, beginning in column number 7.

Using the DISPLAY ... AT ... statement, the user can build up a full screen of information, one item at a time.

4.3.3 Displaying More Complex Screens

When several items are to be displayed, many DISPLAY ... AT ... statements might be required. Declaring FILLER items to fill the intervening gaps simplifies this, thus requiring only one DISPLAY statement.

Example:

To generate

```
SELECT ONE OF THE FOLLOWING ITEMS

    1. FOOTBALL SCORES
    2. TENNIS RESULTS
    3. GOLF NEWS
    4. EXIT
```

the following program could be used for an 80-column screen.

```
ENVIRONMENT DIVISION.
SPECIAL-NAMES.
CONSOLE IS CRT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  DISPLAY-ITEM-1.
    03  DISPLAY-ITEM-1-1  PIC X(33)
        VALUE "SELECT ONE OF THE FOLLOWING ITEMS".
    03  FILLER  PIC X(128).
    03  DISPLAY-ITEM-1-2  PIC X(18)
        VALUE "1. FOOTBALL SCORES".
    03  FILLER  PIC X(62).
    03  DISPLAY-ITEM-1-3  PIC X(17)
        VALUE "2. TENNIS RESULTS".
    03  FILLER  PIC X(63).
    03  DISPLAY-ITEM-1-4  PIC X(12)
        VALUE "3. GOLF NEWS".
    03  FILLER   PIC X(68).
    03  DISPLAY-ITEM-1-5  PIC X(7)
        VALUE "4. EXIT".
        .
        .
        .
PROCEDURE DIVISION.
START-OF-PROGRAM.
        DISPLAY SPACES.
        DISPLAY DISPLAY-ITEM-1 AT 0507.
```

FILLER items are never displayed, even as spaces, so whatever is on the screen before a DISPLAY is still displayed in the places covered by FILLER items.

4.4 Accepting Data Entered at a CRT

After the screen has been set up and displayed, and the user has entered some data, the data must be ACCEPTED.

NOTE

As most terminals scroll upwards as a result of a character appearing in the final (that is, bottom right) character position of the screen, this character position cannot be used as part of an ACCEPT.

Two types of items can be accepted: elementary data items and group items.

4.4.1 Accepting an Elementary Item

The statement

```
ACCEPT MYDATA.
```

places the cursor at the HOME position, and accepts the character string keyed in by the user until terminated by pressing the [RETURN] key. This string is directly transferred into the data item MYDATA, and is aligned left if too short. MYDATA is then checked against its declaration in the DATA DIVISION, and any format errors are reported.

If the AT clause is used, then the value of the data item in the AT clause defines the start position of the ACCEPT data item. This data item must be PIC 9999 where the most significant two digits define a line number in the range 01 to the maximum number of lines on the screen, and the least significant two digits define a column number the range 01 to the maximum number of characters per line on the screen. If data item contains zero or spaces, it is treated as 0101 (HOME). The cursor is positioned at the start of the data item to be accepted; that is, the position defined by the AT clause. See Explicit Cursor Positioning.

Example:

```
ACCEPT MYDATA AT 1021.
```

positions the cursor at column number 21 on line 10, unless the cursor is explicitly placed elsewhere (see Explicit Cursor Positioning), and accepts whatever the user enters.

4.4.2 Accepting a Group Item

Accepting a group item is more complex. The user must declare the group item in the WORKING-STORAGE SECTION of the program. This declaration might be similar to the data declaration used to generate the DISPLAY screen, except that data items in one are probably FILLER fields in the other. In this case, the user might find redefining the original DISPLAY group item as the ACCEPT group item to be advantageous.

Example:

```
ENVIRONMENT DIVISION.
SPECIAL-NAMES.
CONSOLE IS CRT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  DISPLAY-ITEM-1.
    03  FILLER  PIC X(324).
    03  DISPLAY-ITEM-1-1  PIC X(33)
        VALUE "SELECT ONE OF THE FOLLOWING ITEMS".
    03  FILLER  PIC X(128).
    03  DISPLAY-ITEM-1-2  PIC X(18)
        VALUE "1. FOOTBALL SCORES".
    03  FILLER  PIC X(62).
    03  DISPLAY-ITEM-1-3  PIC X(17)
         VALUE "2. TENNIS RESULTS".
    03  FILLER  PIC X(63).
    03  DISPLAY-ITEM-1-4  PIC X(12)
        VALUE "3. GOLF NEWS".
    03  FILLER  PIC X(68).
    03  DISPLAY-ITEM-1-5  PIC X(7)
        VALUE "4. EXIT".
01  ACCEPT-ITEM-1 REDEFINES DISPLAY-ITEM-1.
    03  FILLER  PIC X(504).
    03  ACCEPT-ITEM-1-1  PIC X.
    03  FILLER  PIC X(79).
    03  ACCEPT-ITEM-1-2  PIC X.
    03  FILLER  PIC X(79).
    03  ACCEPT-ITEM-1-3  PIC X.
    03  FILLER  PIC X(79).
    03  ACCEPT-ITEM-1-4  PIC X.

PROCEDURE DIVISION.
START-OF-PROGRAM.
      DISPLAY SPACES.
      DISPLAY DISPLAY-ITEM-1.
      ACCEPT ACCEPT-ITEM-1.
```

In the same manner as DISPLAY ... AT ... , the AT clause can be used to define the initial position of the data, thus avoiding an initial FILLER item in the data declaration (see Explicit Cursor Positioning below). The default position for AT is HOME. HOME is also used if the position defined by AT is outside the physical limits of the screen.

4.4.2.1 Cursor Behavior During an ACCEPT

Unless explicitly positioned by the program (see below), the cursor is initially placed at the start of the first data item to be accepted. While the user is entering data in response to an ACCEPT clause, the cursor advances character by character. If data are entered which do not completely fill the data item, the user must advance the cursor to the next data item by either advancing one space at a time to the end of the current data item, or using the advance-one-field key. The cursor does not move into FILLER items. At the end of the last data item of a group, the cursor remains in the last character position and a bell sounds when any additional character is typed. The last character typed is the one that is accepted.

Data entry to a group item is terminated by pressing the [RETURN] key.

When designing an interactive LEVEL II COBOL program, the user should adopt a consistent approach to ACCEPT statements. A number of individual ACCEPTS on the same screen requires the user to press the [RETURN] key at the end of each one. A group ACCEPT, performing the same function, requires the user to tab forward from field to field (if the fields are not completely filled by the data entered), and press the [RETURN] key only at the end of the last field. Mixing these approaches in any one program or suite of programs might be confusing, and should therefore be avoided.

4.5 Explicit Cursor Positioning

The LEVEL II COBOL user can exercise explicit control over the cursor by using the "CURSOR IS data-name" clause in the SPECIAL-NAMES paragraph. The data name must be a PIC 9999 item, where the most significant two digits define a line number in the range 01 to the maximum number of lines on the screen, and the least significant two digits define a column number in the range 01 to the maximum number of characters per line on the screen.

Example:

```
ENVIRONMENT DIVISION.
SPECIAL-NAMES.
CURSOR IS CURSOR-POSITION.
CONSOLE IS CRT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  CURSOR-POSITION.
    03  CURSOR-LINE PIC 99.
    03  CURSOR-COLUMN PIC 99.
01  DISPLAY-ITEM-1.
    03 FILLER  PIC X (324).
    .
    .
    .
```

On executing an ACCEPT statement, the cursor moves to the character position defined by the CURSOR data item. If the CURSOR data item contains zero or spaces or is undefined, HOME is used by default. Any AT clause in the ACCEPT statement still defines the position of the data items on the screen; the CURSOR data item merely positions the cursor. If the defined position is either outside the physical limits of the screen or outside the limits of the group item or elementary data item being ACCEPTED, the defined position is ignored and the start of the first data item is used instead.

If the defined position is in a FILLER item, the cursor moves to the beginning of the next data item. When no further data item exists, the cursor returns to the beginning of the first data item on the screen.

On return from an ACCEPT statement, the CURSOR data item contains the address of the final position of the cursor on the screen.

One example of this facility is that in menu-type operations the user need only move the cursor to a position on the screen corresponding to the selection required. The user's choice can be determined by the returned value of the CURSOR data item.

If, in this type of operation, one choice per line exists, the resulting line number can be used for a DEPENDING ON clause. The default choice can be determined by explicitly positioning the cursor on one of the choices before the ACCEPT statement.

Note that to use the CURSOR data item for cursor positioning, the data item must contain a value other than zero or spaces. If the CURSOR data item contains zero or spaces, it does not update with cursor positions after ACCEPT statements.

Continuing with the example used earlier in this chapter:

```
SELECT ONE OF THE FOLLOWING ITEMS:

    1. FOOTBALL SCORES
    2. TENNIS RESULTS
    3. GOLF NEWS
    4. EXIT.

POSITION CURSOR AND PRESS RETURN
```

to display the screen and to execute a subroutine depending on the response, the following program could be used.

```
ENVIRONMENT DIVISION.
SPECIAL-NAMES.
CURSOR IS CURSOR-POSITION.
CONSOLE IS CRT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  CURSOR-POSITION.
    03  CURSOR-LINE  PIC 99.
    03  CURSOR-COLUMN  PIC 99.
01  DISPLAY-ITEM-1.
    03  FILLER  PIC X(324).
    03  DISPLAY-ITEM-1-1  PIC X(33)
        VALUE "SELECT ONE OF THE FOLLOWING ITEMS".
    03  FILLER  PIC X(128).
    03  DISPLAY-ITEM-1-2  PIC X(18)
        VALUE "1. FOOTBALL SCORES".
    03  FILLER  PIC X(62).
    03  DISPLAY-ITEM-1-3  PIC X(17)
        VALUE "2. TENNIS RESULTS".
    03  FILLER  PIC X(63).
    03  DISPLAY-ITEM-1-4  PIC X(12)
        VALUE "3. GOLF NEWS".
    03  FILLER  PIC X(68).
    03  DISPLAY-ITEM-1-5  PIC X(7)
        VALUE "4. EXIT".
    03  FILLER  PIC X(153).
    03  DISPLAY-ITEM-1-6  PIC X(32)
        VALUE "POSITION CURSOR AND PRESS RETURN".
01  ACCEPT-ITEM-1 REDEFINES DISPLAY-ITEM-1.
    03  FILLER  PIC X(504).
    03  ACCEPT-ITEM-1-1  PIC X.
    03  FILLER  PIC X(79).
    03  ACCEPT-ITEM-1-2  PIC X.
    03  FILLER  PIC X(79).
    03  ACCEPT-ITEM-1-3  PIC X.
    03  FILLER  PIC X(79).
    03  ACCEPT-ITEM-1-4  PIC X.
PROCEDURE DIVISION.
START-OF-PROGRAM.
        DISPLAY SPACES.
        DISPLAY DISPLAY-ITEM-1.
        MOVE 0625 TO CURSOR-POSITION.
        ACCEPT ACCEPT-ITEM-1.
        SUBTRACT 6 FROM CURSOR-LINE.
        GO TO FOOTBALL-SCORES, TENNIS RESULTS, GOLF NEWS,
        FINISH-OFF DEPENDING ON CURSOR-LINE.
```

CURSOR data-item

DISPLAY item

ACCEPT item

Chapter 5
LEVEL II COBOL Application Design Considerations

LEVEL II COBOL Application Design Considerations

5.1 Introduction

Designing a COBOL application program requires efficient use of the space and facilities available. This chapter is written for designing an application to be written in LEVEL II COBOL, and describes the facilities available:

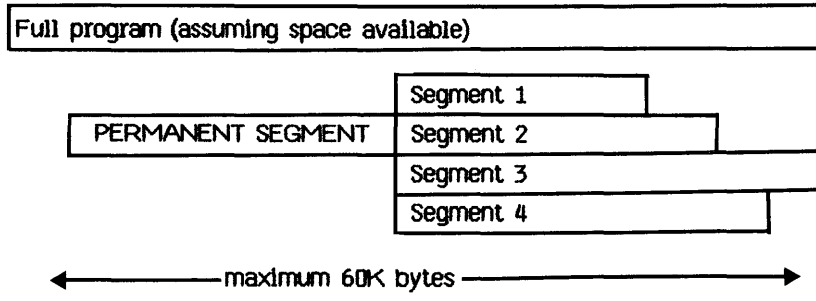
- memory management
- dividing monolithic programs into smaller units
- using default filenames
- calling other programs
- including user-written run-time subroutines
- calling the supplied run-time subroutines.

5.2 LEVEL II COBOL Application Design Facilities

5.2.1 Segmentation (Overlaying)

LEVEL II COBOL enables a COBOL program with a large PROCEDURE DIVISION to be divided into a COBOL program with a small PROCEDURE DIVISION and multiple overlays containing the remainder of the PROCEDURE DIVISION. The resident part is known as the permanent segment and the overlays are known as independent segments.

All of the PROCEDURE DIVISION can be loaded into the available memory by using the LEVEL II COBOL Segmentation feature. However, it cannot be loaded all at once. It is loaded one segment at a time to achieve the same effect in the reduced storage space as shown below.



In a segmented program, the beginning of each segment in the PROCEDURE DIVISION is denoted in the LEVEL II COBOL source code by a SECTION label; for example,

```
.
.
.
SECTION 52.
MOVE A TO B.
.
.
.
SECTION 62.
      MOVE X TO Y.
.
.
.
```

Segmentation can be applied to only the PROCEDURE DIVISION. The IDENTIFICATION, ENVIRONMENT, and DATA DIVISIONs are common to all segments; in addition, a common PROCEDURE DIVISION segment can exist. This common code is known as the permanent segment. Control flow between permanent and independent segments is fully specified in the *COBOL Reference Manual for the Lisa*, Chapter 9.

NOTE

The cumulative size of the DATA DIVISIONs must be less than 64K.

5.2.2 Interprogram Communication (CALL)

LEVEL II COBOL enables COBOL applications to be designed or divided, at source level, into separately compiled programs. Each program is then called dynamically from the main application program, without the need for the user to have linked the programs together first.

Figure 5-1 shows a sample application user interprogram communication.

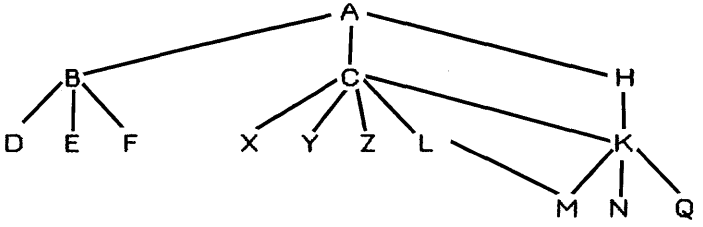


Figure 5-1
Sample CALL Tree Structure

The main program A, which is permanently resident in memory, calls B, C, or H which are subsidiary and standalone functions within the application. These programs call other specific functions as follows:

B calls D, E, and F.
C calls X, Y, or Z conditionally, and K or L conditionally.
H calls K.
K calls M, N, or Q conditionally.
L calls M if it needs to.

As the functions B, C, and H are standalone they do not need to reside permanently in memory together. They can therefore be called as necessary, using the same physical memory when they are called. The same applies to the lower functions at their levels in the tree structure.

In the example shown in Figure 5-1, the use of CALL and CANCEL statements needs to be planned so that a frequently called subroutine, such as K, is kept in memory to save load time. On the other hand, because K is called by C or H, it cannot be called initially without having C or H in memory; that is, the larger of C or H should call K initially to allow space. Avoiding overflow of programs is also important. At the level of X, Y, and Z, the order in which loading takes place does not matter, because calls are not made at a lower level.

Leaving called programs in memory is advantageous if they open files. The EXIT statement does not close files, but the CANCEL statement does. Leaving called programs in memory avoids having to reopen files on every call.

5.2.3 CALL Requirements

Any number of LEVEL II COBOL programs and assembly language routines can be called from a LEVEL II COBOL program. This section describes the requirements of the CALL statement.

1. The CALLed program file must be present on disk both at the time of the first CALL of the program and while the program is being used.
2. Sufficient space must exist in memory for at least the DATA DIVISION to be loaded. The ON OVERFLOW phrase can be used to specify program action to be taken if insufficient space is available, otherwise the CALL statement is ignored and the next program instruction is executed.
3. The CANCEL statement releases the memory occupied by the cancelled program and closes any files opened by it.

If a tree structure of called independent programs as shown earlier is used, each program can call the next dynamically by using the technique shown in the following sample coding:

```
WORKING-STORAGE SECTION.

01  NEXT-PROG      PIC X(20) VALUE SPACES.
01  CURRENT-PROG   PIC X(20) VALUE "STPRG.INT".

PROCEDURE DIVISION.
LOOP.
        CALL CURRENT-PROG USING NEXT-PROG.
        CANCEL CURRENT-PROG.
        IF NEXT-PROG = SPACES STOP RUN.
        MOVE NEXT-PROG TO CURRENT-PROG.
        MOVE SPACES TO NEXT-PROG.
        GO TO LOOP.
```

The actual programs to be run can then specify their successors as follows:

```
.
.
.
LINKAGE-SECTION.
01  NEXT-PROG  PIC X(20).
.
.
.
PROCEDURE DIVISION USING NEXT-PROG.
.
.
.
MOVE "FOLLOW.INT" TO NEXT-PROG.
EXIT PROGRAM.
```

This example demonstrates that each independent segment or subprogram can cancel itself, and, with the USING phrase, change the name in the CALL statement to call the next one.

5.2.4 Producing Compact and Efficient Code

Declaring data items to have usage COMP causes compact storage in the minimum number of bytes needed to accommodate, in binary format, the largest number allowed by the PICTURE string. However, declaring usage COMP does not automatically ensure that arithmetic on such items is efficient as well as compact. Except for the special cases detailed below, arithmetic on COMP data items is done by expansion in internal registers to BCD format, and reconversion to COMP for storing the result.

Efficient coding, known as COMP code, is available for the following types of statements:

1. $\left\{\begin{matrix}\text{ADD}\\\text{SUBTRACT}\end{matrix}\right\}$ source $\left\{\begin{matrix}\text{TO}\\\text{FROM}\end{matrix}\right\}$ target .

   where <u>either</u> both source and target are PIC 9(2) COMP, or both are PIC 9(4) COMP

   <u>or</u> the source is an unsigned integer literal less than 256 and the target is PIC 9(2) COMP, or the source is an unsigned integer literal less than 65536 and the target is PIC 9(4) COMP.

   <u>and</u> there is no ON SIZE ERROR clause.

2. $\left\{\begin{matrix}\text{MULTIPLY}\\\text{DIVIDE}\end{matrix}\right\}$ source $\left\{\begin{matrix}\text{BY}\\\text{INTO}\end{matrix}\right\}$ target .

   where <u>either</u> both source and target are PIC 9(2) COMP, or both are PIC 9(4) COMP.

   <u>and</u> there is no ON SIZE ERROR clause.

   In such cases arithmetic is done on one- or two-byte binary quantities without overflow checking and with binary wraparound.

3. MOVE source to TARGET.

   where the source and target satisfy the rules given above for ADD and SUBTRACT statements.

   In this case the MOVE is a one- or two-byte transfer without data conversion.

4. Comparisons of the form

   left operand relation right operand

   where the operands again satisfy the rules given above for ADD and SUBTRACT statements, except that either (not just the left-hand one, but not both) can be a literal.

   A raw binary one- or two-byte comparison is the result.

5. Finally, even more compact and efficient code is generated for a statement of the form

   IF operand relation literal GO TO label.

where:

the operand is declared as PIC 9(2) COMP and is the first data item in the WORKING-STORAGE SECTION,

the literal is an unsigned integer less than 256, and

no ELSE clause is present.

In Case 4 the efficient code can be generated even when the comparison is just one of a sequence connected by AND/OR. However, Format 5 is totally specific.

Code generated for these statement formats runs more than five times faster than equivalent noncompact code, so taking care to use these formats where possible is worthwhile. However, the interaction between the semantics detailed above and the ANSI COBOL specification must now be examined. The following considerations are relevant:

1. If an ON SIZE ERROR clause is present, the target must not be affected if numeric overflow occurs; COMP code is never generated in such a case. If an ON SIZE ERROR clause is not specified, the result on numeric overflow is implementor defined. In LEVEL II COBOL using COMP code, the result is defined as above; that is, binary byte-oriented arithmetic with wraparound. The user can decide to take advantage of this extra level of definition as a LEVEL II COBOL extension. However, the programs might not then be portable to other ANSI COBOL compilers, because the feature is undefined in ANSI COBOL; alternatively, if the user knows that the arithmetic statements do not lead to numeric overflow, the programs can be portable in any case.

2. When the result of unsigned subtraction is negative, ANSI COBOL requires that the absolute value be stored. COMP code stores the two's complement result. Because of this conflict with ANSI COBOL semantics, COMP code is never generated for SUBTRACT statements unless the user specifies the COMP directive to the compiler; the user should do this either when he or she knows the unsigned COMP subtractions does not underflow (in which case the programs compiled with COMP code remain portable) or when wishing to take advantage of the nonstandard behavior which occurs on underflow.

3. Truncation on MOVE literal: in the statement

```
MOVE literal TO target.
```

where the target is PIC 9(2) COMP and 99<literal<256 or the target is PIC 9(4) COMP and 9999<literal<65536, ANSI COBOL requires that the literal is truncated to the number of decimal places specified for the target. COMP code does not truncate but stores the binary value. As in Case 2 above, because of this conflict the compiler does not generate COMP code for this form of statement unless, for either of the reasons described above, the COMP directive is specified.

Appendixes

Appendix A
Summary of Compiler and Run-Time Directives

A.1 Compiler Directives

The compilation command structure is:

```
LEVEL II COBOL:  Compile, Run, Set Switches, Printer, Quit : C
COBOL Source file [.TEXT] -  filename <<
Compiler directive  -  directive <<
        .    .
        .    .
Compiler directive -  <<
```

Where filename is the name of the file that contains the LEVEL II COBOL source program. The default extension is .TEXT.

A description of the available compiler directives follows.

Directive	Use	Default
[NO] ALTER	Allow ALTER statements	ON
[NO] BRIEF	Suppress error messages	OFF
[NO] COMP	Use computational subset	OFF
[NO] COPYLIST	List COPY files	OFF
CRTWIDTH "n"	Set width of CRT to "n"	ON n = 128
DATE "string"	As DATE below but "string" set to spaces Use "string" for comment entry in DATE-COMPILED paragraph	ON
[NO] ECHO	Echo errors to console	ON
[NO] ERRLIST	List only errors and flags	OFF
[NO] FLAG "[LOW / L-I / H-I / HIGH / L/II / IBM]"	Flag code higher than level indicated	OFF
[NO] FORM "n"	Suppress headers and form-feeds Set length of page = "n" lines	ON n=60
[NO] INT "filename"	Specify intermediate code filename	ON = source filename If no directive: ON; that is filename = source filename. If directive but no filename: filename = -console
NO {LIST / PRINT} "filename"	Specify listing requirements	
[NO] QUAL	Allow qualified data-names and procedure-names	ON
[NO] REF	Insert addresses on listing	OFF
[NO] RESEQ	Resequence source file	OFF

A.2 Run-Time Directives

Before running an L/II COBOL object program the user can modify the program switches (see Section 3.4.1.3 in the *COBOL Reference Manual for the Lisa*) in the range 0-7.

```
LEVEL II COBOL: Compile, Run, Set Switches, Printer, Quit : S

Current Switch settings

    SW-0  SW-1  SW-2  SW-3  SW-4  SW-5  SW-6  SW-7
    OFF   OFF   OFF   OFF   OFF   OFF   OFF   OFF

Do you wish to change the settings?: y

    SWITCH SW:0 = OFF .FLIP? y
    SWITCH SW:1 = OFF .FLIP? n
    SWITCH SW:2 = OFF .FLIP? n
    SWITCH SW:3 = OFF .FLIP? y
    SWITCH SW:4 = OFF .FLIP? y
    SWITCH SW:5 = OFF .FLIP? n
    SWITCH SW:6 = OFF .FLIP? n
    SWITCH SW:7 = OFF .FLIP? n

Updated Switch settings

    SW-0  SW-1  SW-2  SW-3  SW-4  SW-5  SW-6  SW-7
    ON    OFF   OFF   ON    ON    OFF   OFF   OFF

Current ANSI Debug IS switched OFF

Do you wish to change the setting?: y

    Updated ANSI Debug IS switched ON

LEVEL II COBOL: Compile, Run, Set Switches, Printer, Quit :
```

The switches remain in this state until remodified or until the COBOL environment is left.

Appendix B
Compile-Time Errors

Listed below are the error descriptions that correspond to the error numbers printed on listings produced by the LEVEL II COBOL compiler. In case of alternative meanings, relevancy is obvious from context.

ERROR	DESCRIPTION
01	Compiler Error
02	Illegal format: data name
03	Illegal format: literal
04	Illegal format: character
05	Data name not unique
06	Too many data or procedure names declared
07	Obligatory reserved word missing
08	Nested COPY statement or unknown COPY file specified
09	'.' missing
10	Statement starts in the wrong area of the source line
21	'.' missing
22	DIVISION missing
23	SECTION missing
24	IDENTIFICATION missing
25	PROGRAM-ID missing
26	AUTHOR missing
27	INSTALLATION missing
28	DATE-WRITTEN missing
29	SECURITY missing
30	ENVIRONMENT missing
31	CONFIGURATION missing
32	SOURCE-COMPUTER missing
33	MEMORY SIZE/COLLATING SEQUENCE/SPECIAL-NAMES clause in error
34	OBJECT-COMPUTER missing
36	SPECIAL-NAMES missing
37	SWITCH Clause error or system name/mnemonic name error
38	DECIMAL-POINT Clause error
39	CONSOLE Clause error
40	Illegal currency symbol
41	'.' missing
42	DIVISION missing
43	SECTION missing

ERROR	DESCRIPTION
44	INPUT-OUTPUT missing
45	FILE-CONTROL missing
46	ASSIGN missing
47	SEQUENTIAL or INDEXED or RELATIVE missing
48	ACCESS missing on indexed/relative file
49	SEQUENTIAL/DYNAMIC missing or >64 alternate keys
50	Illegal ORGANIZATION/ACCESS/KEY combination
51	Unrecognized phrase in SELECT Clause
52	RERUN Clause syntax error
53	SAME AREA Clause syntax error
54	missing or illegal file name
55	DATA DIVISION missing
56	PROCEDURE DIVISION missing or unknown statement
57	program collating sequence not defined *
58	EXCLUSIVE, AUTOMATIC, or MANUAL missing *
59	Nonexclusive lock mode specified for restricted file
62	DIVISION missing
63	SECTION missing
64	file name not specified in SELECT stmt or invalid CD name
65	RECORD SIZE integer missing or line sequential rec > 1024 bytes
66	Illegal level no, (01-49), or 01 level req'd, or level hierarchy wrong
67	FD, CD, or SD qualification syntax error
68	WORKING-STORAGE missing
69	PROCEDURE DIVISION missing or unknown statement
70	Data description qualifier or '.' missing
71	Incompatible PICTURE clause and qualifiers
72	BLANK illegal with nonnumeric data item
73	PICTURE clause too long
74	VALUE clause with nonelementary item, wrong data type, or value truncated
75	VALUE in error or illegal for PICTURE type
76	nonelementary FILLER/SYNC/JUSTIFIED/BLANK clause
77	Preceding item at this level has > 8192 bytes or 0 bytes
78	REDEFINES of unequal fields or different levels.
79	Data storage exceeds 64K bytes
81	Data description qualifier inappropriate or repeated
82	REDEFINES data name not declared
83	USAGE must be COMP, DISPLAY, or INDEX
84	SIGN must be LEADING or TRAILING
85	SYNCHRONIZED must be LEFT or RIGHT
86	JUSTIFIED must be RIGHT
87	BLANK must be ZERO
88	OCCURS must be numeric, nonzero, unsigned, or DEPENDING

ERROR	DESCRIPTION
89	VALUE must be a literal, numeric literal, or figurative constant
90	PICTURE string has illegal precedence or illegal char
91	INDEXED data name missing or already declared
92	Numeric-edited PICTURE string is too large
101	Unrecognized verb
102	IF ... ELSE mismatch
103	operand has wrong data type or is not declared
104	Procedure name not unique
105	Procedure name same as data name
106	Name required
107	Wrong combination of data types
108	Conditional statement not allowed in this context
109	Malformed subscript
110	ACCEPT/DISPLAY wrong or Communications syntax incorrect
111	Illegal Syntax used with I/O verb
112	Invalid arithmetic statement
113	Invalid arithmetic expression
114	PROCEDURE DIVISION in memory > 32K
115	Invalid conditional expression
116	IF statements nested too deep or too many AFTERs in PERFORM stmt
117	Incorrect structure of PROCEDURE DIVISION
118	Reserved Word missing or incorrectly used
119	Too many subscripts in one statement
120	Too many operands in one statement *
121	LOCK clause specified for EXCLUSIVE file *
122	KEPT specified for uncommitable file *
123	KEPT omitted for commitable file
141	Intersegment procedure name duplication
142	IF ... ELSE mismatch at end of Source Input
143	operand has wrong data type or not declared
144	Procedure name undeclared
145	Index data name declared twice
146	Bad cursor control: illegal AT clause
147	KEY declaration missing or illegal
148	STATUS declaration missing
149	Bad STATUS record
150	Undefined intersegment reference or error in ALTERed par
151	PROCEDURE DIVISION in error
152	USING parameter not declared in LINKAGE SECTION
153	USING parameter is not level 01 or 77
154	USING parameter used twice in parameter list
155	FD missing

ERROR	DESCRIPTION
157	Incorrect structure of PROCEDURE DIVISION
160	Too many operands in one statement

* - Apply to Fileshare optional product syntax.

In addition to these numbered error messages, the following message can be displayed with subsequent termination of the compilation:

I-O ERROR: { filename / OBJECT FILE }

where filename is the erroneous file.

OBJECT FILE is one of .INT, .D??, or.I?? (for segmented programs)

Any intermediate code file produced is not usable.

The following conditions causes this error:

Disk overflow
File directory overflow
File full
Impossible I/O device usage

Other Operating System dependent conditions can also cause this error.

The error numbers in the preceding list are not continuous: those that are not listed are be produced only if an error occurs in the compiler. When this happens, contact Technical Support immediately.

Appendix C
Run-Time Errors

Run-Time Errors

C.1 Error Reporting

Two types of run-time errors exist: Recoverable and Fatal.

C.1.1 Recoverable Errors

File handling errors (codes 0 - 99) do not cause termination of program execution if the programmer has specified a STATUS field for the file concerned. In this case, the RTS returns the character '9' in Status Key 1 of the STATUS field, and the COBOL RTS error code, in binary (COMP), in Status Key 2 field. (Because this error code is stored in binary (COMP) in Status Key 2, the only way you can extract it is with the method shown in the sample error handling routine, Section C.3.) The RTS takes no other action: the user must check for specific error conditions and take corrective action, or terminate the program run.

If the programmer has not specified STATUS on that file, any file handling error is a fatal error.

C.1.2 Fatal Errors

When the RTS detects a fatal error, the general class of error, along with its associated file name, is printed out. The RTS then prints out the error code, the COBOL program counter (pc), the CALL number, and segment number corresponding to the statement where the error occurred.

Fatal errors fall into two categories:

1. Exceptions are errors detected by the RTS, such as arithmetic overflow, subscripts out of range, and INT. file load errors. These error codes are in the range 100-200 decimal. A typical message appears:

   ```
   Load error:  file 'MYPROG.153'
   RTS error code:  160, pc=0085, call=0, seg=53
   Consult COBOL User's Guide
   ```

   The program is terminated, and control returns to the COBOL command line.

2. File Handling Errors are errors in the range 0-99 decimal, on a user file for which STATUS was not specified. For these errors, the original error message signaled from the OS is displayed, along with the corresponding RTS error code that would have been stored in Status Key 2 if a STATUS field had been specified. A typical message appears:

```
OS error message =     921
I/O error:  file '-MYPROG>DAT'
RTS error code: 12, pc=0085, call=0, seg=0
Consult COBOL User's Guide
```

   The program is terminated, and control returns to the COBOL command line.

C.2 Run-Time Error Codes

The run-time error codes and their meanings are:

C.2.1 File Errors

1	Unspecified error on attempting to OPEN file
2	Unspecified error on attempting to WRITE to file
3	Unspeicifed error on attempting to ACCEPT FROM CRT
4	Unspecified error on attempting to READ file
5	Unspecified error on attempting to WRITE to line sequential file
6	Unspecified error on attempting to CLOSE file
12	Invalid pathname for file, or no such device
13	File not found
14	Unexpected file system error
15	No space available on disk for creating or extending file
16	Access denied by OS for specified operation on file; for example, file locked
17	Too many files open: attempt to OPEN more files (20) than system allows

C.2.2 Exceptions

151	Random READ on a sequential file
152	Attempt to REWRITE on a file not opened I/O
153	Subscript bounds overflow; for example, zero, or greater than defined range
154	PERFORM nesting exceeds allowed limit of 55 levels
155	Illegal command line
156	Invalid file operation
157	INT. file too large: not enough program memory for loading it
158	REWRITE on a line sequential file
159	Malformed line sequential file
160	Overlay loading error; for example, file not found, or invalid file structure
161	Illegal intermediate code: program file probably corrupt
162	Arithmetic overflow or underflow

164 Specified CALL subroutine not supplied
165 Incompatible version of Compiler and RTS: recompile source program
166 Attempt to OPEN a file that is already open
167 Attempt to CLOSE a file not already open
170 Illegal operation in Indexed Sequential module
172 Recursive CALL illegal; for example, attempting to CALL an active program
173 Intermediate code file not found
174 Intersegment reference file for a segmented program cannot be loaded
180 COBOL file malformed
181 Fatal file malformation
182 Attempt to OPEN :CI: or :CO: in illegal direction
183 Attempt to OPEN a line sequential file for I/O
184 ACCEPT/DISPLAY error
185 Cannot load COBOL RTS module; for example, IXSIO.INT
186 Internal RTS error: contact Technoical Support
188 File name too long
189 Intermediate code load error
190 Too many arguments to CALLed subprogram
191 Terminal type not defined
192 Required terminal capability not supported
193 Null file name used in a file operation
194 Memory allocation error

C.3 Sample Error Handling Routine

Note that the original 2-character status field must be redefined as a (COMP) field, and that LOW-VALUES must be moved to the Status Key 1 field before the error code can be displayed or printed.

```
000010 ENVIRONMENT DIVISION.                                    0118
000020 INPUT-OUTPUT SECTION.                                    0118
000030 FILE-CONTROL.                                            0118
000040 SELECT FILE1 ASSIGN"TST.FIL"                             0184
000050   STATUS IS FILE1-STAT.                                  0186
000060 DATA DIVISION.                                           01BD
000070 FILE SECTION.                                            01BD
000080 FD  FILE1.                                               01BD
000090 01  F1-REC  PIC X(80).                                   01BD
000100 WORKING-STORAGE SECTION.                                 020F
000110 01  FILE1-STAT.                                          020F 00
000120     02  S1  PIC X.                                       020F 00
000130     02  S2  PIC X.                                       0210 01
000140 01  STAT-BIN REDEFINES FILE1-STAT  PIC(4) COMP.          020F 00
000150 01  DISPLAY-STAT.                                        0211 02
000160     02  S1-DISPL  PIC X.                                 0211 02
000170     02  FILLER  PIC X(3).                                0212 03
000180     02  S2-DISPL  PIC 9999.                              0215 06
000190 PROCEDURE DIVISION.                                      0000
000200         OPEN INPUT FILE1.                               001A
000210         IF S1 NOT = 9  GO TO PARA1.                     001E
000220                                                         0030
000230         MOVE S1 TO S1-DISPL.                            0030
000240         MOVE LOW-VALUES TO S1.                          0035
000250         MOVE STAT-BIN TO S2-DISPL.                      003A
000260         DISPLAY DISPLAY-STAT.                           0041
000270 PARA1.                                                  004C 00
000280         STOP RUN.                                       004D
```

Appendix D
File Formats

File Formats

D.1 Fixed File Assignment

D.1.1 ENVIRONMENT DIVISION

In the FILE-CONTROL Paragraph, the general format of the SELECT and ASSIGN TO statements follows.

D.1.1.1 General Format

```
SELECT    file-name
      ASSIGN TO        external-file-name-literal.
                       [ , external-file-name-literal] .
```

D.1.1.2 Parameters

file-name	is any user-defined LEVEL II COBOL word.
external-file-name-literal	is a standard Lisa file name, enclosed in quotes, of one the following general formats.

Format 1

```
-device
```

where device is a logical device as follows:

```
PRINTER     - line printer
CONSOLE     - screen output
KEYBOARD    - keyboard input
```

Example:

```
-printer
```

Format 2 (a Lisa pathname)

[-volumename-]filename[.extension]

where:

volumename is the name that the user has assigned to a volume or a physical, block-structured device as follows:

UPPER
LOWER
PARAPORT
SLOTmCHANn

Files on the working directory can be specified without volumename and the delimiting "-".

filename is between one and 32 alphabetic or numeric characters, spaces are permitted. The combined length of the filename, plus extension, plus the '.' delimiting the extension must not exceed 32 characters.

extension is up to 32 alphabetic or numeric characters, spaces are permitted, including the delimiting '.'. However, the same restriction as in filename applies.

Example:

[-paraport-]myprog[.text]

Format 3

-device-dummy

where:

device is the name of a physical, sequential device as follows:

RS232A
RS232B.

dummy is a dummy file name.

Example:

-RS232A-X

D.1.1.3 Example

```
SELECT  STOCKFILE
        ASSIGN TO "WAREHS.BUY".
```

D.1.2 PROCEDURE DIVISION

The file name specified above is then specified in the OPEN statement when the file is required for use in the program. See The OPEN Statement in Chapters 5, 6, and 7 of the *COBOL Reference Manual for the Lisa.*

D.2 Run-Time File Assignment

The internal user file name is assigned to a file identifier, an alphanumeric user-defined COBOL word, which automatically sets up a PIC X(66) data area in which to store the external Lisa file name; the user can specify a differently sized data area by explicitly declaring it. The user can then store the external Lisa file name in this data area in the PROCEDURE DIVISION, and can alter it during the run as required.

The following specifications are required for run time assignment:

D.2.1 ENVIRONMENT DIVISION

In the FILE-CONTROL Paragraph the general format of the SELECT and ASSIGN TO statements is as follows:

D.2.1.1 General Format

SELECT file-name

ASSIGN TO file-identifier.

D.2.1.2 Parameters

file-name	is any user-defined L/II COBOL word.
file-identifier	is any other user-defined L/II COBOL word

D.2.1.3 Example

```
SELECT STOCK-FILE
     ASSIGN TO STOCK-NAME.
```

D.2.2 PROCEDURE DIVISION

Before the file is OPENed for use, the external Lisa file name of the required file (see Fixed File Assignment above for format) is stored, as required in the file identifier location specified above by the user program.

Example:

```
MOVE    "WAREHS.BUY" TO STOCK-NAME.
OPEN    INPUT    STOCK-FILE.
.
.
.
CLOSE   STOCK-FILE.
.
.
.
MOVE    "WAREHS.SEL" TO STOCK-NAME.
OPEN    INPUT    STOCK-FILE.
.
.
.
CLOSE   STOCK-FILE.
```

The Lisa file name can be entered via an ACCEPT statement, by an user, or stored as any other variable data.

In this way, a different external file can be used as a common internal user file during any run of a program, but care is required to ensure that the correct file is allocated at any given time.

Note that once the OPEN statement has been executed, the file identifier data area can be used for any purpose the user requests. In the above example, between the two OPEN sentences, STOCK-NAME can be used for storing any data string required.

D.3 LEVEL II COBOL Disk File Structures under the Lisa

LEVEL II COBOL offers four types of file organization for use by the COBOL user: sequential, line sequential, relative, and indexed sequential (ISAM). A file is a set of records. A record is a set of contiguous data bytes which are mapped into hardware sectors with which they need not coincide; that is, a record can start anywhere within a sector and can span hardware sector boundaries. The data are held as follows:

D.3.1 Sequential

Sequential files are read and written using fixed length records. The length used is that of the longest record defined in the COBOL program's FD.

Normally the space occupied per record is the same as the program record length, and data of any type can be held on the file. However, this does not apply if WRITE statements are specified using BEFORE or AFTER ADVANCING phrases. Then extra control characters are inserted, and the data can no longer be read back correctly.

No limits exist on file size beyond those imposed by the Operating System and/or hardware.

D.3.1.1 Sequential File Format

A sequential file consists of fixed length records without terminating characters. The file therefore appears as a string of characters. If the last record in the file contains trailing 1AH characters, it might not be accessible.

D.3.2 Line Sequential

Line sequential file format is intended for text (ASCII) files generated by editors and other similar utilities, and is the only LEVEL II COBOL file format that supports variable length records. The one-byte 0D carriage return is used as a record delimiter. On input, the 0D is removed and the record area filled with spaces as necessary. On output, any trailing spaces in the program's record area are ignored. Using ADVANCING phrases other than BEFORE 1 causes the output of additional device control characters; a file created this way can be read by a program. If a record is too long; that is, it exceeds the maximum length specified for the file, each access returns 'maximum-length' characters until the end of the record. After an access, if the next two characters are 0D, they are omitted -- in this case, a blank line is not be returned on the next access.

D.3.3 Relative

Relative file organization provides a means of accessing data randomly by specifying its position in the file. Records are of fixed length. The length used is that of the longest record defined in the program's FD. To designate whether or not a record logically exists, one byte is added to the end of each record: this byte is 0D if the record logically exists on the file and 00 if it does not. The total length of a file is determined by the highest relative record number used; LEVEL II COBOL imposes no effective limit on this value. Data of any type can be held on the file.

D.3.3.1 Relative File Format

A relative file is organized similarly to a sequential file; it is a string of characters in fixed length records. However, each record is followed by a one-byte interrecord marker which is:

carriage return	Hex 0D	if the preceding record exists, or
null	Hex 00	if the preceding record doesn't exist.

If a record is deleted from a relative file, the interrecord marker following the record is changed to Hex 00 - the state of the data is undefined. If, for security purposes, the user wishes to be certain the data are deleted, the record should be overwritten by using a COBOL REWRITE statement before deletion.

A relative file can be read in sequential mode, if the record length is set to Relative-record-length+1.

Note that the first record in a relative file is record one: no record zero exists.

D.3.4 Indexed Sequential

The Indexed Sequential Access Method (ISAM) uses two types of files: the data file and the key or index file. Both types of files are in relative file format.

The name of the data file is supplied by the user. The name of the associated key file is produced by adding the extension .IDX to the root of the data file name.

Example:

Data file	Key file
MYFILE	MYFILE.IDX
CLOCK.FLE	CLOCK.IDX

NOTE

Two or more data files with different extensions, but the same root name, cannot be distinguished by ISAM as the key files all have the same name. To avoid using the extension .IDX in other contexts is also advisable.

The index is built up as an inverted tree structure which grows in height as records are added. The number of key file accesses required to locate a randomly selected record depends primarily on the number of records on the file and the 'keylengths'. An approximate guide to the number of levels in the tree, and hence the number of accesses required, is

$$\text{index levels} \quad \log_k \quad \text{(number of records)}$$

$$\text{where } k \quad \simeq \quad \frac{150}{\text{keylength} + 2}$$

but varies slightly on the order in which records are added and deleted.

Faster response times are obtainable when reading a file sequentially, but only if other ISAM operations do not intervene.

The size, in bytes, of an ISAM file is approximately related to the maximum number of records it contains as follows:

data = (record length + 2) * max. no of records

$$\text{index} = \frac{\text{no of records}}{k - 1} * 256 \quad \text{where k is as defined above}$$

NOTE

The necessity of taking regular backup copies of all types of files cannot be emphasized too strongly, and should always be regarded as the main safeguard. However, situations exist with indexed files; for example, media corruption, that can lead to only one of the two files becoming unusable. If the index file is lost in this way, it is normally possible to recover data records from just the data file, although not in key sequence, and cut down on the time lost due to a failure. As an aid to this, all unused data records are marked as deleted at the relative file level by appending two bytes to each record that contains LOW-VALUES. For undeleted records, these bytes contain the characters carriage return and line feed. The recovery operation can therefore be done with a simple COBOL program, by defining the data file as ORGANIZATION SEQUENTIAL ACCESS SEQUENTIAL with records defined as two bytes longer than in the ISAM file description. The records are then read sequentially, and the data MOVEd from the sequential file record area into the indexed (ISAM) file record area and written to a new version of the indexed file; except for those records with LOW-VALUES in the last two (extra) bytes, these records should be discarded. Note that these two bytes, containing carriage return and line feed characters in a required record, are not written to the ISAM file on recovery, because of the record length discrepancy of two bytes in the record definitions.

Appendix E
Useful Facts and Figures

E.1 PERFORM Nesting

Due to the mechanism used for controlling the nesting of PERFORM statements, LEVEL II COBOL allows a maximum of 55 nested PERFORMS.

The PERFORM stack is between 470 and 480 bytes in size.

This depth of PERFORM nesting applies to a whole CALLed suite of programs; any PERFORMs active at a CALL remain active.

E.2 USING Parameters

The maximum number of PROCEDURE DIVISION USING parameters is 12.

E.3 Level 01 Entries

There can be up to 63 Level 01 entries in the LINKAGE SECTION. This limit does not include redefinitions.

E.4 Exponentiation

Fractional exponents are rounded down to the nearest positive integer before evaluation. Negative exponents are correctly evaluated, but not fractions. Exponents can be up to four digits in length. If larger than 9999, the overflow flag is set.

E.5 Size of Numbers

According to the ANSI standard:

> Numbers are limited to 18 significant decimal digits.
> All significant digits are within 18 digits of the decimal point.

In LEVEL II COBOL the result of a multiplication or division that is greater than 36 digits gives a SIZE ERROR, as does the result of an addition or subtraction that is greater than 37 digits.

E.6 Open Files

The maximum number of files that a COBOL program can have open simultaneously is 20 files. Exceeding this causes the RTS to signal Error 17 (see Appendix C).

Appendix F
COBOL Workshop Files

COBOL Workshop Files

This appendix lists the files on the COBOL 1.0 diskettes.

File Name	COBOL Diskette	Notes	Description
BYE.TEXT	1		Workshop installation exec file.
ByteDiff.obj	2		Utility program.
Cistart.text	1		Workshop installation exec file.
COBOL.ERR	2		Workshop program.
COBOL.I51	2		Workshop program.
COBOL.I52	2		Workshop program.
COBOL.I53	2		Workshop program.
COBOL.I56	2		Workshop program.
COBOL.I59	2		Workshop program.
COBOL.INT	2		Workshop program.
COBOL.ISR	2		Workshop program.
COBOL.OBJ	2		Workshop program.
Diff.obj	2		Utility program.
DumpPatch.obj	2		Utility program.
EDIT.MENUS.TEXT	2		Editor support file.
Editor.obj	2		Workshop program.
Filediv.obj	2		Utility program.
Filejoin.obj	2		Utility program.
find.obj	2		Utility program.
FMDATA	1	1,2	Data segment.
font.heur	1	1,2,3	Data needed to support SYS1Lib.
FONT.HEUR	2		Second copy of same file.
font.lib	1	1,2,3	Data needed to support SYS1Lib.
GETPROFILELOC.TEXT	1		Workshop installation exec file.
GETYESNO.TEXT	1		Workshop installation exec file.
INSERTDISK.TEXT	1		Workshop installation exec file.
Intrinsic.lib	1	2,3	Library directory.
IOSFplib.obj	2		Library unit w/interface.
IOSPaslib.obj	1	2,3	Library unit w/interface.
IXSIO.INT	2		Workshop program.

Note 1: These files are identical to Office System Release 1.0 files.

Note 2: These files are identical to Office System Release 1.2 files. Office System 1.2 is functionally identical to Office System 1.0, but is released to ensure compatibility with Pascal 1.0, BASIC-Plus 1.0, and COBOL 1.0.

Note 3: These files are the minimum necessary to run a user program in the Workshop environment. A user program may require other files as well.

File Name	COBOL Diskette	Notes	Description
LDSPREFERENCES.OBJ	2		Workshop program.
LDS_RES_PROCS.TEXT	2		Workshop data.
OSERRS.ERR	1	3	Workshop data.
PAPER.TEXT	2		Workshop data.
PI.TEXT	2		COBOL demonstration program.
Portconfig.obj	2		Utility program.
resident_channel	1	1,2,3	System data.
Shell.WorkShop	1	3	Workshop main program.
STOCK1.TEXT	2		COBOL demonstration program.
STOCK2.TEXT	2		COBOL demonstration program.
Sulib.obj	1	3	Library unit w/interface.
Sxref.obj	2		Utility program.
SXREF.OMIT.TEXT	2		Data.
Syslib.obj	1	1,2,3	Library units (no interface).
SYS2LIB.OBJ	2	1,2,3	Library units (no interface).
SYSTEM.BT_PROF	1	1,2,3	System support.
SYSTEM.BT_TWIG	1	1,2,3	System support.
SYSTEM.IUDIRECTORY	1	1,2,3	System data.
SYSTEM.LLD	1	1,2,3	System program.
SYSTEM.LOG	1	1,2,3	System data.
SYSTEM.OS	1	2,3	System program.
System.Shell	1	1,2,3	System program.
SYSTEM.STACK1	1	1,2,3	System data.
SYSTEM.STACK2	1	1,2,3	System data.
SYSTEM.STACK3	1	1,2,3	System data.
SYSTEM.STACK4	1	1,2,3	System data.
SYSTEM.SYSLOC1	1	1,2,3	System data.
SYSTEM.SYSLOC2	1	1,2,3	System data.
SYSTEM.SYSLOC3	1	1,2,3	System data.
SYSTEM.SYSLOC4	1	1,2,3	System data.
SYSTEM.TIMER_PIPE	1	1,2,3	System data.
SYSTEM.UNPACK	1	1,2,3	System data.
term.menus.text	2		Data for transfer program.
transfer.obj	2		Workshop program.
WMDATA	1	1,2	Data segment.
{T11}BUTTONS	2	2	Data.
{T11}MENUS.TEXT	2	2	Data.

Note 1: These files are identical to Office System Release 1.0 files.

Note 2: These files are identical to Office System Release 1.2 files. Office System 1.2 is functionally identical to Office System 1.0, but is released to ensure compatibility with Pascal 1.0, BASIC-Plus 1.0, and COBOL 1.0.

Note 3: These files are the minimum necessary to run a user program in the Workshop environment. A user program may require other files as well.

Index

Index

Please note that the topic references in the Index are *by section number.*

THIS MANUAL was produced using LisaWrite, LisaDraw, and LisaList.

ALL PRINTING was done with an Apple Dot Matrix Printer.

the Lisa™

...we use it ourselves.

Apple publications would like to learn about readers and what you think about this manual in order to make better manuals in the future. Please fill out this form, or write all over it, and send it to us. We promise to read it.

How are you using this manual?
[ ] learning to use the product [ ] reference [ ] both reference and learning

[ ] other_____

Is it quick and easy to find the information you need in this manual?
[ ] always [ ] often [ ] sometimes [ ] seldom [ ] never

Comments_____

What makes this manual easy to use?_____

_____

What makes this manual hard to use?_____

_____

What do you like most about the manual?_____

_____

What do you like least about the manual?_____

_____

Please comment on, for example, accuracy, level of detail, number and usefulness of examples, length or brevity of explanation, style, use of graphics, usefulness of the index, organization, suitability to your particular needs, readability.

_____

_____

_____

What languages do you use on your Lisa? (check each)

[ ] Pascal [ ] BASIC [ ] COBOL [ ] other_____

How long have you been programming?

[ ] 0-1 years [ ] 1-3 [ ] 4-7 [ ] over 7 [ ] not a programmer

What is your job title?_____

Have you completed:

[ ] high school [ ] some college [ ] BA/BS [ ] MA/MS [ ] more

What magazines do you read?_____

_____

_____

Other comments (please attach more sheets if necessary)_____

_____

_____

_____

FOLD

FOLD

PLACE
STAMP
HERE

POS Publications Department
20525 Mariani Avenue
Cupertino, California 95014

TAPE OR STAPLE

COBOL for the Lisa

Release 1.0 Notes

What's in the COBOL Release Notes?

These notes describe situations that were brought to our attention after it was too late to document them in the COBOL manuals.

Insert these notes in the back of their respective manuals, so that you can refer to them as necessary.

If you have a question or a problem that you can't find the answer to, either in the manuals or in these notes, you should call the Lisa Telephone Support Line, (800) 553-4000.

029-0454-A

November 1983

Manual Chapter	Release Note
Workshop Chapter 1	The installation instructions state that you must install the Lisa Office System before installing any optional language products. However, these instructions apply to only the installation order. You do not need to install the Office System if you intend to do only language development.
Workshop Chapter 1	After you press the on-off button (at the start of installation), wait for a quick tone before selecting the disk drive.
Workshop Chapter 1	Although the installation instructions state that the installation procedure should be aborted if any error messages are returned, you might normally encounter error 950 or 948 when you try to install SYSTEM.TIMER_PIPE and RESIDENT_CHANNEL. You might also encounter error 1176 for these pipes if you use the Equal command after installation.
Workshop Chapter 1	Correctly installing COBOL 1.0 on top of your Office System Release 1.0 pulls the Office System up to level 1.2. All subsequent installations of system software are then order dependent, requiring installation from the Workshop to follow that of the Office System. *Do not* reinstall the Office I and Office II diskettes without immediately reinstalling the language products(s). However, if your Office System is already at level 1.2, the installation is order independent.
Workshop Chapter 1	After successfully adding COBOL to a ProFile containing the Office System, if the system is merely allowed to reboot, the default of the Environments window will cause the Office System to restart. To cause the initialization to pause at the Environments window in order to examine or change the default, press the space bar after the machine selftest, while the hourglass icon is showing.
Workshop Chapter 1	If you have just printed anything on a daisy wheel printer from the Office System, and you return to the Workshop using the Environments window, printing to logical device "-printer" will be garbled until the printer is switched off and then on again.

Manual Chapter	Release Note
Workshop Chapter 1	The print commands of the Editor always use the logical device "-printer" set in the System Manager. Choosing "Daisy Wheel Printer" or "Dot Matrix Printer" from the Print menu does not change the System's configuration, but only adjusts the Editor to the intended device.
Workshop Chapter 1	Any program intended to run as a background process (MakeBackgroundProcess) must include frequent and judicious calls to the Operating System procedure Yield_CPU. Hence, system utilities should never be run in the background. Also, a background process should not have any interaction with the console, and it cannot pull events from the hardware event queue.
Workshop Chapter 2	Designating user files to begin with the pathname "SHELL." makes them appear in the Environments window as an alternative shell.
Workshop Chapter 2	You cannot directly rename a file to a name that differs from the original only in the case of the characters, because the internal representation of the names is the same. Instead, rename the file to a temporary name, and then change that to the name you want.
Workshop Chapter 2	If you unmount the prefix volume by ejecting the diskette, Scavenging the volume, or using the Unmount command, the boot volume automatically becomes the prefix volume.
Workshop Chapter 3	The Output Redirect function of the System Manager does not correctly handle screen output that uses GOTOXY, for example, screen output done by the File Manager when listing wildcard matches. This results in redirected output to the printer being overwritten on one line.
Workshop Chapter 4	The Editor changes the creation date of a text file to the current date each time the file is modified.

Manual Chapter	Release Note
Workshop Chapter 4	If the initialization of the Editor fails due to lack of disk space (error 309), and space on the disk is then made free, the next attempt to start the Editor will also fail (error 304). You must enter the Process Manager of the System Manager, Kill the Editor process, and then retry.
Workshop Chapter 4	The language processors, Editor, and many other utilities of the Workshop expect as input a standard .TEXT file. The internal structure of a text file in a block-structured device is described in the Lisa COBOL Reference Manual: • Each page (two 512-byte blocks) contains some number of complete lines of text and is padded with null characters (ASCII 0) after the last line as necessary to complete the page. • Two 512-byte header blocks are also present at the beginning of the file. These may or may not contain information. • A sequence of spaces (ASCII 32 decimal, $20 hexadecimal) can be compressed into a 2-byte code namely, a DLE character (ASCII 16 decimal, $10 hexadecimal), followed by a byte containing the value 32 decimal plus the number of spaces represented.
Workshop Chapter 4	The file name "PAPER.TEXT" is reserved for the default stationery template of the Editor and should not be used for other purposes.
Workshop Chapters 4 and 10	Cursor residue might be left on the screen in the Editor and the Transfer program, especially after an error message has appeared.
Workshop Chapters 4 and 10	The names of files created by the Editor and Transfer will be changed to be all upper case, regardless of how they are typed in.
Workshop Chapter 7	If multiple errors occur during a link, due an attempt to link regular units with intrinsic units, the Linker will terminate after reporting only the first error.

Manual Chapter	Release Note
Workshop Chapter 8	For the Debugger, >PR 2 is print to SLOT2CHAN2, not SLOT2CHAN1. Upper and lower are reversed in the manual.
Workshop Chapter 10	Display of error message 647 while you are using the Transfer utility might indicate only that after a timeout the program has failed to receive the appropriate handshake from the host.
Workshop Chapter 10	If you type any key during "Playback from what file " in the Transfer program, the playback will abort.
Workshop Chapter 10	If you use the Transfer program to make contact with a host computer, and you exit the program without logging off explicitly, the connection will not be automatically terminated. This is usually a convenience, but might not meet user expectations.
Workshop Chapter 10	When the Workshop shell is initialized, all serial ports are configured by default as if they were printers (e.g., 9600 baud, DTR handshake, automatic linefeed insertion), whether or not they are listed as such by Preferences. If you subsequently use and then exit the Transfer program, the printer configuration is restored automatically for ONLY those ports listed in Preferences as printers; others will retain the properties set by the Transfer program. The Editor will not reconfigure ports that have been changed by PortConfig.
Workshop Chapter 10	To terminate recording to a file opened by the Transfer program during "Record to", open the Control menu and again select "Record to". This terminates recording and closes the file. Note that, unlike the Editor, Transfer does not automatically insert a carriage return at the end of the file. If you use this recording to capture text such as a source program, and the language processor (such as BASIC-Plus) expects to see a carriage return at the end of the file, attempting to run the raw recorded text might cause the system to hang.
Workshop Chapter 10	The manual states that the default handshake in the Transfer program is XOn/XOff. The correct default is None.

Manual Chapter	Release Note
Workshop Appendix B	ASCII characters in the range hex 20 through hex 7E are supported for screen display, for printing on a dot matrix printer, and for printing on a daisy wheel printer with the following print wheels: • Gothic, 15 pitch. • Prestige Elite, 12 pitch. • Courier, 10 pitch. • Boldface/Executive, PS. Printing ASCII characters to a daisy wheel printer is not supported for the three print wheels with Modern type styles. The character set in the Appendix should show the full Lisa Character Set. All of the additional characters can be displayed on the screen. Most additional characters can be printed on a dot matrix printer, but none will print on a daisy wheel printer. A new page B-1 is attached; take a moment now to make the substitution.

Manual Chapter	Release Note
COBOL User's Guide Chapter 1	An easy way to adapt the Mouse Editor for COBOL programs is to use the Edit menu to set tabs every eight spaces, tab to the 72nd space, and resize the window to show only 72 columns of text.
COBOL User's Guide Chapter 2	To avoid destroying a valid .INT file when recompiling source files with the same name, you can use the compiler directive NO INT or INT "SOMETHING.INT".
COBOL User's Guide Chapter 2	An error during compilation sometimes results in a file ending in .D00 being left on the disk. Ignore or delete the .D00 file.
COBOL Reference Chapter 8	User-defined collating sequences produce unpredictable results in some cases.

Appendix B
Workshop Character Set

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SP	0	@	P	`	p	Ä	ê	†	∞	¿			
1	SOH	DC1	!	1	A	Q	a	q	Å	ë	°	±	¡			
2	STX	DC2	"	2	B	R	b	r	Ç	í	¢	≤	¬			
3	ETX	DC3	#	3	C	S	c	s	É	ì	£	≥	√			
4	EOT	DC4	$	4	D	T	d	t	Ñ	î	§	¥	ƒ			
5	ENQ	NAK	%	5	E	U	e	u	Ö	ï	•	µ	≈			
6	ACK	SYN	&	6	F	V	f	v	Ü	ñ	¶	∂	Δ			
7	BEL	ETB	'	7	G	W	g	w	á	ó	ß	∑	«			
8	BS	CAN	(	8	H	X	h	x	à	ò	®	∏	»			
9	HT	EM	)	9	I	Y	i	y	â	ô	©	π	...			
A	LF	SUB	*	:	J	Z	j	z	ä	ö	™	∫				
B	VT	ESC	+	;	K	[	k	{	ã	õ	´	ª				
C	FF	FS	,	<	L	\	l	|	å	ú	¨	º			
D	CR	GS	-	=	M	]	m	}	ç	ù	≠	Ω				
E	SO	RS	.	>	N	^	n	~	é	û	Æ	æ				
F	SI	US	/	?	O	_	o	DEL	è	ü	Ø	ø				

The first 32 characters and DEL are nonprinting control codes.

The shaded area is reserved for future use.

COBOL

Packing List

This package contains the following items:

Item	Quantity	Part Number	Description
1	1	620-6137	COBOL Reference Manual for the Lisa™
2	1	682-0015	COBOL 1, Diskette
3	1	682-0053	COBOL 2, Diskette
4	1	620-6148	Workshop User's Guide for the Lisa™
5	1	620-6155	COBOL User's Guide for the Lisa™
6	1	029-0183	Software Registration

In case of questions, contact the dealer from whom you purchased this product.