APPLE
PROGRAMMER'S
AND DEVELOPER'S
ASSOCIATION

290 SW 43rd. Street
Renton, WA 98055
206-251-6548

Lisa Workshop Supplement

Version 1.0

APDA# KMSLW1

About the Lisa Workshop Supplement

Disks in the Lisa Workshop Supplement

The Lisa Workshop Supplement contains the following three disks:

Lisa Workshop 1	Lisa Workshop 3.0 formatted disk; can be used with any version of the Lisa Workshop from version 3.0 on. Contains equate files needed when using the Lisa Workshop to create Macintosh applications.
Lisa Workshop 2	Lisa Workshop 3.0 formatted disk; can be used with any version of the Lisa Workshop from verison 3.0 on. Contains object files needed when using the Lisa Workshop to create Macintosh applications.
Lisa Workshop 3	Lisa Workshop 3.0 formatted disk; can be used with any verison of the Lisa Workshop from version 3.0 on. Contains interface files needed when using the Lisa Workshop to create Macintosh applications. Also contains the Resource File Builder, described in Section 7 of this document.
Lisa Update 1,2	Lisa Workshop 3.0 formatted disks; can be used with any version of the Lisa Workshop from version 3.0 on. Contain the necessary files to update the Lisa Workshop from version 3.0 to version 3.9. Includes new versions of the Workshop Shell, the linker, the editor, the assemble, RMaker, and MacCom.

Please note: The files included here are meant to **replace** all of the previous versions that were sent out in previous Software Supplements. Please **do no use your older versions;** use these newer copies.

Contents of the Lisa Workshop Supplement Disks:

Contents of Lisa Workshop 1:

```
TLAsm/ATalkEqu.text
TLAsm/Fixmath.text
TLAsm/FSEqu.text
TLAsm/FSPrivate.text
TLAsm/Graf3DEqu.text
TLAsm/HardwareEqu.text
TLAsm/PackMacs.text
TLAsm/PREqu.text
TLAsm/Private.text
TLAsm/QuickEqu.text
TLAsm/SANEMacs.text
TLAsm/SCSIEqu.text
TLAsm/SonyEqu.text
TLAsm/SysEqu.text
TLAsm/SysErr.text
TLAsm/TimeEqu.text
TLAsm/ToolEqu.text
TLAsm/Traps.text
```

Contents of Lisa Workshop 2:

```
ATalk/ABPackage.obj
ATalk/ABPackage.Rsrc
ATalk/ABPackageR.text
obj/ABPasCalls.obj
obj/ABPasIntf.obj
obj/AppleTalk.obj
obj/FixMath.obj
obj/FixMathAsm.obj
obj/Graf3D.obj
obj/Graf3DAsm.obj
obj/MacPrint.obj
obj/MemTypes.obj
obj/OSIntf.obj
obj/OSTraps.obj
obj/PackIntf.obj
obj/PackTraps.obj
obj/PasInit.obj
obj/PasLib.obj
obj/PasLibAsm.obj
obj/PasLibIntf.obj
obj/PrintCalls.obj
obj/QuickDraw.obj
obj/ResEd.obj
obj/RTLib.obj
obj/SANELib.obj
obj/SANELibAsm.obj
obj/SCSIIntf.obj
obj/SCSITraps.obj
obj/ToolIntf.obj
```

```
obj/ToolTraps.obj
obj/WritelnWindow.obj
serial/Async/Mac.obj
serial/Async/MacXL.obj
serial/AsyncR.text
```

Contents of Lisa Workshop 3:

```
intrfc/AppleTalk.text
intrfc/FixMath.text
intrfc/Graf3D.text
intrfc/MacPrint.text
intrfc/MemTypes.text
intrfc/OSIntf.text
intrfc/PackIntf.text
intrfc/PasLibIntf.text
intrfc/QuickDraw.text
intrfc/ResEd.text
intrfc/SANELib.text
intrfc/SCSIIntf.text
intrfc/ToolIntf.text
intrfc/WritelnWindow.text
RFB.obj
RFB/exec.text
SOURCE/RFB.TEXT
```

Documents in the Lisa Workshop Supplement:

Putting Together A Macintosh Application describes the steps for creating an application using the Lisa Workshop.

The **Lisa Workshop Update** tells how to update your verison 3.0 Lisa Workshop to version 3.9. If you have version 2.0, you must update to version 3.0 before you can use this update.

The **Writeln Window** is a debugging aid that you can use in your Lisa Workshop Pascal programs to help in development of applications.

The document **Equate and Glue Files for the Lisa Workshop** explains the object and text files used for development under the Lisa Workshop.

PPostEvent, NGetTrapAddress and NSetTrapAddress detail new Trap call which have been added.

The printout **Yanked Text** lists equates which have been removed from public and private include files. The list can be searched for routines, equates, or other structures which your program may have referenced in the past, but are no longer resolved due to their removal from the include files.

The **Resource File Builder** is a utility that aids in application development where lots of resources are used. It can save time by preventing recompilation of RMaker files at each build.

Putting Together a Macintosh Application

/PUTTING/TOGETHER

Modification History:	First Draft (ROM 2.45)	Caroline Rose	6/9/83
	Second Draft (ROM 4.4)	Caroline Rose	7/14/83
	Third Draft (ROM 7)	Caroline Rose	1/13/84
	Fourth Draft	Caroline Rose	4/9/84
	Fifth Draft	Caroline Rose	7/10/84
	Sixth Draft	Caroline Rose	5/5/85

ABSTRACT

This manual discusses the fundamentals of preparing, compiling or assembling, and linking a Macintosh application program on the Lisa Workshop development system.

Summary of significant changes and additions since last draft:

-This manual now documents Lisa Workshop version 3.0 and the May 1985 Macintosh Software Supplement. Some of the information may not apply to Workshop version 2.0.

-Changes have been made to the interface files and the files you link with or include in your assembly-language source.

-The sections describing the Macintosh utility programs RMover and Set File have been removed. These programs have been superseded by other tools in the Macintosh Software Supplement.

CONTENTS

ABOUT THIS MANUAL

This manual discusses the fundamentals of preparing, compiling or assembling, and linking a Macintosh application program on the Lisa Workshop development system. It assumes the following:

-You know how to write a Macintosh application in Pascal or assembly language. Details on this may be found in Inside Macintosh.

-You're familiar with the Macintosh Finder, which is described in Macintosh, the owner's guide.

You need to have a Lisa 2/5 or 2/10 with at least 1 megabyte of memory, a Workshop development system (version 2.0 or greater), and the Macintosh Software Supplement.

Note: This manual applies to version 3.0 of the Workshop and the May 1985 Software Supplement.

After explaining some conventions it uses, the manual begins by presenting the first steps you should take once your Lisa has been set up for Macintosh application development under the Workshop. It then discusses each of the three files you'll create to develop your application: the source file, the Resource Compiler input file, and an exec file.

The next section discusses how to divide an application into segments. This is followed by important information for programmers who want to write all or part of an application in assembly language.

Finally, there's a summary of the steps to take to put together a Macintosh application.

Note: This manual presents a recommended scenario, not by any means the only possible one. Details, such as what you name your files, may vary.

Conventions

Sometimes this manual shows you what to do in a two-column table, the first one labeled "Prompt" and the second "Response". The first column shows what appears on the Lisa to "prompt" you; it might be a request for a file name, or just the Workshop command line. This column will not show all the output you'll get from a program, only the line that prompts you. (There may have been a lot of output before that line.) The second column shows what you type as a response. The following notation is used:

Notation	Meaning
<ret>	Press the RETURN key.
[ ]	Explanatory comments are enclosed in [ ]; you don't type them.

A space preceding <ret> is not to be typed. It's there only for readability.

[ ] in the "Prompt" column actually appear in the prompt; they

enclose defaults.

Except where indicated otherwise, you may type letters in any combination of uppercase and lowercase, regardless of how they're shown in this manual.

GETTING STARTED

Once your Lisa has been set up for Macintosh application development, it's a good idea to orient yourself to the files installed on it. You can use the List command in the File Manager to list all the file names. Certain subsets of related files begin with the same few letters followed by a slash; some typical naming conventions are as follows:

Beginning of file name	Description
Intrfc/	Text files containing the Pascal interfaces
TlAsm/	Text files to include when using assembly language
Obj/	Object files
Work/	Your current working files
Back/	Backup copies of your working files
Example/	Examples provided by Macintosh Technical Support

Note: This manual assumes that your files observe the above naming conventions.

You'll write your application to a Macintosh system disk, which means a Macintosh disk that contains the system files needed for running an application. The necessary system files are on the Mac Build disk that you received as part of the Macintosh Software Supplement. Use that disk only to create other system disks. Here's how:

1. Insert the Mac Build disk into the Macintosh and open it.

2. Copy the System Folder to a new Macintosh disk; the exact method you use depends on whether you have an external drive. See the Macintosh owner's guide for more information.

Note: One of the files in the System Folder, Imagewriter, is needed only if you're going to print to an Imagewriter printer; to save space, you might not want to copy it if you don't need it.

If you also need or want any of the files on the MacStuff disks included in the Macintosh Software Supplement, copy them as well.

As described in detail in the following sections, you'll create a source file, Resource Compiler input file, and exec file for your application, insert your Macintosh system disk into the Lisa, and run the exec file. The exec file will compile the source file, link the resulting object file with other required object files, run the Resource Compiler to create the application's resource file, and run a program called MacCom to write the application to the Macintosh disk. When MacCom is done, it will eject the disk; to try out your application, you'll insert the ejected disk into the Macintosh and just open the application's icon.

THE SOURCE FILE

Your working files will of course include the source file for your application. Suppose, for example, that you have an application named Samp. The source file would be Work/Samp.Text and would have the structure shown below.

Note: "Samp" is used as the application name in all examples in this manual. You don't have to use the exact name of your application; anyabbreviation will do.

```
PROGRAM Samp;

{ Samp -- A sample application written in Pascal }
{         by Macintosh User Education  5/1/85     }

[ List the following in the order shown. ]

USES {$U Obj/MemTypes    } MemTypes,
     {$U Obj/QuickDraw   } QuickDraw,
     {$U Obj/OSIntf      } OSIntf,
     {$U Obj/ToolIntf    } ToolIntf,
     {$U Obj/MacPrint    } MacPrint,     [ OPTIONAL ]
     {$U Obj/SANELib     } SANELib,      [ OPTIONAL ]
     {$U Obj/PackIntf    } PackIntf;     [ OPTIONAL ]

[ Your LABEL, CONST, TYPE, and VAR declarations will be here. ]

[ Your application's procedures and functions will be here. ]

BEGIN

  [ The main program will be here. ]

END.
```

Each line in the USES clause specifies first a file name and then a unit name (which happen to be the same in all cases here). The file contains the compiled Pascal interface for that unit; the corresponding text file name begins with "Intrfc/" rather than "Obj/". The Pascal interface includes the declarations of all the routines in the unit. It also contains any data types, predefined constants, and, in the case of QuickDraw, Pascal global variables.

File name	Interface it contains
Intrfc/MemTypes.Text	Basic Memory Manager data types
Intrfc/QuickDraw.Text	QuickDraw
Intrfc/OSIntf.Text	Operating System
Intrfc/ToolIntf.Text	Toolbox, except QuickDraw
Intrfc/MacPrint.Text	Printing Manager
Intrfc/SANELib.Text	Floating-Point Arithmetic and

	Transcendental Functions Packages
Intrfc/PackIntf.Text	Other packages

You only have to include the files for the units your application uses. It doesn't do any harm to include them all, but it will take somewhat longer for your program to compile. If you're using any units of your own, just add their Pascal interface files at the end of the USES clause.

You can divide the code of an application into several segments and have only some of them in memory at a time. The section "Dividing Your Application Into Segments" tells how to specify segments in your source file. If you don't specify any, your program will consist of a single, blank named segment.

THE RESOURCE COMPILER INPUT FILE

You'll need to create a resource file for your application. This is done with the Resource Compiler, and you'll have among your working files an input file to the Resource Compiler. One convention for naming this input file is to give it the name of your source file followed by "R" (such as Work/SampR.Text).

The first entry in the input file specifies the name to be given to the output file from the Resource Compiler, the resource file itself; you'll enter "Work/" followed by the application name and ".Rsrc". Another entry tells which file the application code segments are to be read from. (The code segments are actually resources of the application.) You'll enter the name of the Linker output file specified in the exec file for building your application, as described in the next section.

If you don't want to include any resources other than the code segments, you can have a simple input file like this:

```
*  SampR -- Resource input for sample application
*           Written by Macintosh User Education  5/1/85

Work/Samp.Rsrc

Type SAMP = STR
  ,0
Samp Version 1.1 -- May 1, 1985

Type CODE
  Work/SampL,0
```

This tells the Resource Compiler to write the resulting resource file to Work/Samp.Rsrc and to read the application code segments from Work/SampL.Obj. It also specifies the file's signature and version data, which the Finder needs.

It's a good idea to begin the input file with a comment that describes its contents and shows its author, creation date, and other such information. Any line beginning with an asterisk (*) is treated as a comment and ignored. (You cannot have comments embedded within lines.) The Resource Compiler also ignores the following:

-leading spaces (except before the text of a string resource)

-embedded spaces (except in file names, titles, or other text strings)

-blank lines (except for those indicated as required)

The first line that isn't ignored specifies the name to be given to the resulting resource file. Then, for each type of resource to be defined, there are one or more "Type statements". A Type statement consists of the word "Type" followed by the resource type (without quotes) and, below that, an entry of following format for each resource:

```
  file name!resource name,resource ID (resource attributes)
  type-specific data
```

The punctuation shown here in the first line is typed as part of the format. Don't enter spaces where none are shown, such as after the comma. You must always provide a resource ID. Specifications other than the resource ID may or may not be required, depending on the resource type:

-Either there will be some type-specific data defining the resource or you'll give a file name indicating where the resource will be read from. Even in the absence of a file name, you QmustR include the comma before the resource ID.

-You specify a resource name along with the file name for fonts and drivers. The Menu Manager procedures AddResMenu and InsertResMenu will put these resource names in menus. Enter the names in the combination of uppercase and lowercase that you want to appear in the menus.

-Resource attributes in parentheses are optional for all types. They're given as a number equal to the value of the resource attributes byte, and 0 is assumed if none is specified. For example, for a resource that's purgeable but has no other attributes set, the input will be "(32)".

If you want to enter a nonprinting or other unusual character in your input file, either by itself or embedded within text, just type a back slash (\) followed by the ASCII code of the character in hexadecimal. For example, the Resource Compiler interprets \0D as a Return character and \14 as the apple symbol.

The formats for the different types of resources are best explained by example. Some examples are given below along with remarks that provide further explanation. Here are some points to remember:

-Most examples list only one resource per Type statement, but you can include as many resources as you like in a single statement.

-In every case, resource attributes in parentheses may be specified after the resource ID.

-All numbers are base 10 except where hexadecimal is indicated.

-The Type statements may appear in any order in the input file.

```
Type WIND                     Window template
  ,128                        Resource ID
  Status Report               Window title
  40 80 120 300               BoundsRect (top left bottom right)
  Visible GoAway              For FALSE, use Invisible or NoGoAway
  0                           ProcID (window definition ID)
  0                           RefCon (reference value)

Type MENU                     Menu, standard type
  ,128                        Resource ID (becomes the menu ID)
* menu for desk accessories
  \14                         Menu title (apple symbol)
    About Samp...             Menu item
                              Blank line required at end of menu
  ,129                        Resource ID
  Edit                        Menu title
    Cut/X                     Menu items, one per line, with meta-
    Paste/Z                    characters, ! alone for check mark
    (-                        You cannot specify a blank item; use (-
    Word Wrap!                 for a disabled continuous line.
                              Blank line required at end of menu
Type MENU                     Menu, nonstandard type
  ,200                        Resource ID   [ SEE NOTE 1 BELOW ]
  201                         Resource ID of menu definition procedure
  Patterns                    Menu title (may be followed by items)
                              Blank line required at end of menu
Type CNTL                     Control template
  ,128                        Resource ID
  Help                        Control title
  55 20 75 90                 BoundsRect
  Visible                     For FALSE, use Invisible
  0                           ProcID (control definition ID)
  1                           RefCon (reference value)
  0 0 0                       Value minimum maximum

Type ALRT                     Alert template
  ,128                        Resource ID
  120 100 190 250             BoundsRect
  300                         Resource ID of item list
  F721                        Stages word in hexadecimal

Type DLOG                     Dialog template
  ,128                        Resource ID
* modal dialog
  100 100 190 250             BoundsRect
  Visible 1 NoGoAway 0        1 is procID, 0 is refCon
  200                         Resource ID of item list
                              Title (none in this case)
  ,129
* modeless dialog
  100 100 190 250             BoundsRect
  Visible 0 GoAway 0          0 procID, 0 refCon
  300                         Resource ID of item list
  Find and Replace            Title
```

```
Type DITL                     Item list in dialog or alert
  ,200                        Resource ID
  5                           Number of items
  BtnItem Enabled             Also:  ChkItem, RadioItem
  60 10 80 70                 Display rectangle
  Start                       Title
                              Blank line required between items
  ResCItem Enabled            Control defined in control template
  60 30 80 100                Display rectangle
  128                         Resource ID of control template

  StatText Disabled           Also:  EditText
  10 93 26 130                Display rectangle
  Seed                        The text (may be blank if EditText)

  IconItem Disabled           Also:  PicItem
  10 24 42 56                 Display rectangle
  128                         Resource ID of icon

  UserItem Disabled           Application-defined item
  20 50 60 85                 Display rectangle

Type ICON                     Icon
  ,128                        Resource ID
  0380 0000                   The icon in hexadecimal (32 such lines
    . . .                      altogether)
  1EC0 3180

Type ICN#                     Icon list
  ,128                        Resource ID
  2                           Number of icons
  0001 0000                   The icons in hexadecimal (32 such lines
    . . .                      altogether for each icon)
  0002 8000

Type CURS                     Cursor
  ,300                        Resource ID
  7FFC . . . 287F             The data:  64 hex digits on one line
  0FC0 . . . 1FF8             The mask:  64 hex digits on one line
  0008 0008                   The hotSpot in hexadecimal (v h)

Type PAT                      Pattern
  ,200                        Resource ID
  AADDAA66AADDAA66            The pattern in hexadecimal

Type PAT#                     Pattern list
  ,136                        Resource ID
  2                           Number of patterns
  5522552255225522            The patterns in hexadecimal, one per
  FFEEDDCCFFEEDDCC             line

Type STR                      String
  ,128                        Resource ID
This is your string           The string on one line (leading spaces
                               not ignored)
Type STR#                     String list
  ,129                        Resource ID
First string                  The strings
```

```
Second string
* note Return in next string
Third string\0Dcontinued
                                   Blank line required after last string
Type DRVR                          Desk accessory or other device driver
  Obj/Monkey!Monkey,17 (32)        File name!resource name,resource ID
                                    [ SEE NOTE 2 BELOW ]
Type FREF                          File reference
  ,128                             Resource ID
  APPL  0  TgFil                   File type  local ID of icon  file name
                                    (omit file name if none)
Type BNDL                          Bundle
  ,128                             Resource ID
  SAMP  0                          Bundle owner
  2                                Number of types in bundle
  ICN#  1                          Type and number of resources
  0  128                           Local ID 0 maps to resource ID 128
  FREF  1                          Type and number of resources
  0  128                           Local ID 0 maps to resource ID 128

Type FONT                          Font (or FWID for font widths)
  Obj/Griffin!Griffin,400@0        File name!resource name,resource ID
  Obj/Griffin10,400@10             File name,resource ID   [ SEE NOTE 3 ]
  Obj/Griffin12,400@12             File name,resource ID   [ BELOW      ]

Type CODE                          Application code segments
  Obj/SampL,0                      Linker output file name,resource ID
                                    [ SEE NOTE 4 BELOW ]
```

Notes:

1. Notice that the input for a nonstandard menu has one extra line in it: the resource ID of the menu definition procedure, just following the resource ID of the menu. If that line is omitted (that is, if the menu's resource ID is followed by a line containing text rather than a number), the resource ID of the standard menu definition procedure (0) is assumed.

2. The Resource Compiler adds a NUL character (ASCII code 0) at the beginning of the name you specify for a 'DRVR' type of resource. This inclusion of a nonprinting character avoids conflict with file names that are the same as the names of desk accessories.

3. The resource ID for a font resource has a special format:

```
  font number @ size
```

The actual resource ID that the Resource Compiler assigns to the font is

```
  (128 * font number) + size
```

Three font resources are listed in the example above. Size 0 is used to provide only the name of the font (Griffin in this case); a file name must also be specified but is ignored. The two remaining font resources define the Griffin font in two sizes, 10 and 12.

4. For a 'CODE' type of resource, ".Obj" is appended to the given file name, and the resource ID you specify is ignored. The Resource Compiler

always creates two resources of this type, with ID numbers 0 and 1, and will create additional ones numbered sequentially from 2 if your program is divided into segments.

The Type statement for a resource of type 'WDEF', 'MDEF', 'CDEF', 'FKEY', 'KEYC', 'PACK', or 'PICT' has the same format as for 'CODE': Only a file name and a resource ID are specified. For the 'PICT' type, the file contains the picture; for the other types, it contains the compiled code of the resource, and the Resource Compiler appends ".Obj" to the file name.

Note: The 'MBAR' resource type is not recognized by the Resource Compiler.

If your application is going to write to the resulting resource file as well as read it, you should place the Type statement for the code segments at the end of the input file. In general, any resources that the application might change and write out to the resource file should be listed first in the input file, and any resources that won't be changed (like the code segments) should be listed last. The reason for this is that the Resource Compiler stores resources in the reverse of the order that they're listed, and it's more efficient for the Resource Manager to do file compaction if the changed resources are at the end of the resource file.

Defining Your Own Resource Types

You can use one of the three types GNRL, HEXA, and ANYB to define your own types of resources in the Resource Compiler input file. GNRL allows you to specify your resource data in the manner best suited to your particular data format; you specify the data as you want it to appear in the resource. A code (beginning with a period) tells the Resource Compiler how to interpret what you enter on the next line or lines (up to the next code or the end of the Type statement). The following illustrates all the codes:

```
Type GNRL                    General type
  ,128                       Resource ID
  .P                         Pascal strings (with length byte), one
A Pascal string               per line
Another Pascal string
  .S                         Strings without length byte, one per
A string                      line
  .I                         Integers (decimal), one per line
  0
  1
  .L                         Long integers (decimal), one per line
  5438
  .H                         Bytes in hexadecimal, any number
  526FEEC942E78EA4            total, any number per line
  0F4C
  .B                         Bytes from a file
  MyData  36  256            File name  number of bytes  offset
                             Blank line required at end of statement
```

You can use an equal sign (=) along with the GNRL type to define a resource of any desired format and with any four-character resource type; for example, to define a resource of type 'MINE' consisting of the integer 57 followed by the Pascal string 'Finance charges', you could enter this:

```
Type MINE = GNRL
  ,400
  .I
  57
  .P
Finance charges
```

The Resource Manager call GetResource('MINE',400) would return a handle to this resource.

The types HEXA and ANYB simply offer alternatives to the .H and .B options (respectively) of the GNRL type, as shown below.

```
Type HEXA                      Bytes in hexadecimal
  ,201                         Resource ID
  526FEEC942E78EA4             The bytes (any number total, any
  0F4C                          number per line)
                               Blank line required at end

Type ANYB                      Bytes from a file
  MyData,200                   File name,resource ID
  36  256                      Number of bytes  offset in file
```

You can also define a new resource type that inherits the properties of a standard type. For example,

```
  Type XDEF = WDEF
```

defines the new type 'XDEF', which the Resource Compiler treats exactly like 'WDEF'. The next line would contain a file name and resource ID just as for a 'WDEF' resource.

THE EXEC FILE

It's useful for each application to have an exec file that does everything necessary to build the application, including compiling, linking, creating the resource file, and writing to a Macintosh disk. The name of the exec file might, for example, be the source file name followed by "X" (for "eXec"). Work/SampX.Text, the exec file for the Samp application, is shown below.

```
$EXEC
P{ascal}$M+
Work/Samp
{no list file}
{default output file}
L{ink}?
```

```
+X
{no more options}
Work/Samp
Obj/QuickDraw
Obj/OSTraps
Obj/ToolTraps
Obj/PrLink          [ OPTIONAL ]
Obj/SANELibAsm      [ OPTIONAL ]
Obj/PackTraps       [ OPTIONAL ]
Obj/PasInit
Obj/PasLib
Obj/PasLibAsm
Obj/RTLib
{end of input files}
{listing to console}
Work/SampL
R{un}RMaker
Work/SampR
R{un}MacCom
F{inder info}Y{es}L{isa->Mac}Work/Samp.Rsrc
Samp
APPL
SAMP
{no bundle bit}
E{ject}Q{uit}
$ENDEXEC
```

The file begins with $EXEC and ends with $ENDEXEC. Everything in between (except for comments in braces) is exactly what you would type on your Lisa if you were not using an exec file. To show what the various entries in this file accomplish, the table below indicates what each of them is a response to, and shows your response as it is in the exec file or as it would be if you were using the keyboard. The numbers on the left are given for reference in the explanation that follows the table.

	Prompt	Response
[1]	Workshop command line	P [for Pascal]
	Input file - [.TEXT]	Work/Samp <ret>
	List file - [.TEXT]	<ret> [for none]
	Output file - [Work/Samp][.OBJ]	<ret> [for Work/Samp.Obj]
[2]	Workshop command line	L [for Link]
	Input file [.OBJ] ?	?~<ret> [for options]
	Options ?	+X <ret>
	Options ?	<ret> [no more options]
	Input file [.OBJ] ?	Work/Samp <ret>
	Input file [.OBJ] ?	Obj/QuickDraw <ret>
	Input file [.OBJ] ?	Obj/OSTraps <ret>
	. . .	[other input files]
	Input file [.OBJ] ?	Obj/RTLib <ret>
	Input file [.OBJ] ?	<ret> [end of input files]
	Listing file [-CONSOLE] / [.TEXT]	<ret> [for -CONSOLE]
	Output file ? [OBJ.]	Work/SampL <ret>
[3]	Workshop command line	R [for Run]

```
     Run what program?                       RMaker <ret>
     Input file [sysResDef][.TEXT] -         Work/SampR <ret>

[4]  Workshop command line                   R  [for Run]
     Run what program?                       MacCom <ret>
     MacCom command line                     F  [for Finder info]
     Always prompt for the Finder info
      when writing to a Mac file?
      (Y or N) [No]                          Y  [for Yes]
     MacCom command line                     L  [for Lisa->Mac]
     Lisa files to write to Mac disk?        Work/Samp.Rsrc <ret>
     Copy to what Mac file?                  Samp <ret>
     Type? [????]                            APPL <ret>
     Creator? [????]                         SAMP <ret>
     Set the Bundle Bit? (Y or N) [No]       <ret>  [for No]
     MacCom command line                     E  [for Eject]
     MacCom command line                     Q  [for Quit]
```

Here's what you accomplish at each of the steps:

1. You compile the Pascal source code (Work/Samp.Text), resulting in an objectfile (Work/Samp.Obj).

2. You link the application's object file with other object files (resulting in the output file Work/SampL.Obj).

3. You run the Resource Compiler to create the application's resource file (Work/Samp.Rsrc, as specified in Work/SampR.Text, the input file to the Resource Compiler). Included in the resources are the application's code segments, which are read from the Linker output file.

4. You use the MacCom program to write the resource file to the Macintosh disk, giving the file the exact name you want your application to have. You set its file type to 'APPL' and its creator to the signature specified in the resource file. Since there's no bundle in Samp's resource file, you don't set the bundle bit. Finally, you ask MacCom to eject the disk.

The files linked with the application's object file in step 3 are described below. Most of them contain a trap interface, which is a set of small assembly language routines that make it possible to call the corresponding unit or units from Pascal. The files should be listed in the order shown. Specify the optional files only if your application uses the routines they apply to.

File name	Description
Obj/MemTypes.Obj	Basic Memory Manager data types
Obj/QuickDraw.Obj	Pascal interface to QuickDraw, needed so the Linker will know how many QuickDraw globals there are
Obj/OSTraps.Obj	Trap interface for the Operating System
Obj/ToolTraps.Obj	Trap interface for the Toolbox (except QuickDraw)
Obj/PrLink.Obj	The Printing Manager (except low-level)

```
Obj/PrScreen.Obj        The low-level Printing Manager routines;
                        can be specified Qinstead ofR PrLink
Obj/SANELibAsm.Obj      The Floating-Point Arithmetic and
                        Transcendental Functions Packages
Obj/PackTraps.Obj       Trap interface for other packages

Obj/PasInit.Obj         \
Obj/PasLib.Obj           \ Predefined Pascal routines,
Obj/PasLibAsm.Obj        / such as POINTER and ORD4
Obj/RTLib.Obj           /
```

Before running the Exec file, insert a Macintosh system disk into the Lisa. Run the exec file as follows:

Prompt	Response
Workshop command line	R [for Run]
Run what program?	<Work/SampX <ret>

When the disk is ejected, remove it and insert it into the Macintosh. To try out your application, just open its icon.

Warning: If you don't set your application's file type and creator, either you won't be able to open its icon in the usual way, or a different application may start up when you do open it!

Notice that if you change the application's signature or the setting of its bundle bit, step 4 of the above exec file will have to be edited accordingly. Furthermore, if you create an icon for your application (or modify it), you'll have to delete the invisible Desktop file, otherwise the Finder won't know about the new icon. You can delete the Desktop file by using the Delete command in MacCom on the Lisa, just before copying the application to the disk with MacCom, or by holding down the Option and Command keys when you start up the system disk on the Macintosh.

Note: Deleting the Desktop file can also affect the folder structure on the disk.

Before making major changes to your application, it's a good idea to back it up. You can use the Backup command in the File Manager to back up all files beginning with "Work/" to files beginning with "Back/" (Work/=,Back/=). Also, you might want to periodically back up your working files onto 3~1/2-inch disks.

There are several ways you could refine the exec file illustrated here; exactly what you do will depend on your particular situation. Some possibilities are listed below.

-You can set up the exec file to compile or link only if actually necessary. For more information, see your Workshop documentation or the sample general purpose exec file (Example/Exec.Text) provided in the Macintosh Software Supplement.

-To save disk space, you can add commands to the exec file to make it delete the two intermediate files: the object file for the application and the Linker output file.

-If you want to keep the intermediate files around but are working on more than one application, you can save disk space by giving the intermediate files the same name for all applications (say, "Work/Temp").

-You can embed the exec file in your program's source file. To do this, you must use "(*" and "*)" around the exec part of the file and use the I invocation option. See your Workshop documentation for details.

DIVIDING YOUR APPLICATION INTO SEGMENTS

You can specify the beginning of a segment in your application's source file as follows:

```
  {$S segname}
```

where segname is the segment name, a sequence of up to eight characters. Normally you should give the main segment a blank name. For example, you might structure your program as follows:

```
PROGRAM Samp;

[ The USES clause and your LABEL, CONST, and VAR declarations
  will be here. ]

{$S Seg1}

[ The procedures and functions in Seg1 will be here. ]

{$S Seg2}

[ The procedures and functions in Seg2 will be here. ]

{$S      }

BEGIN

  [ The main program will be here. ]

END.
```

You can specify the same segment name more than once; the routines will just be accumulated into that segment. To avoid problems when moving routines around in the source file, some programmers follow the practice of putting a segment name specification before every routine.

Warning: Uppercase and lowercase letters QareR distinguished in segment names. For example, "Seg1" and "SEG1" are not equivalent names.

If you don't specify a segment name before the first routine in your file, the blank segment name will be assumed there.

NOTES FOR ASSEMBLY-LANGUAGE PROGRAMMERS

You can write all or part of your Macintosh application in assembly language. Suppose, for example, that you write most of it in Pascal but have some utility routines written in assembly language. Your working files will include a source file and object file for the assembly-language routines (say, Work/SampA.Text and Work/SampA.Obj). The source file will have the structure shown below.

```
; SampA -- Assembly-language routines for Samp
;          Written by Macintosh User Education  5/1/85

[ List the following in the order shown. ]

.INCLUDE TlAsm/SysEqu.Text
.INCLUDE TlAsm/SysTraps.Text
.INCLUDE TlAsm/SysErr.Text
.INCLUDE TlAsm/QuickEqu.Text
.INCLUDE TlAsm/QuickTraps.Text
.INCLUDE TlAsm/ToolTraps.Text
.INCLUDE TlAsm/ToolEqu.Text
.INCLUDE TlAsm/PrEqu.Text        [ OPTIONAL ]
.INCLUDE TlAsm/SANEMacs.Text     [ OPTIONAL ]
.INCLUDE TlAsm/PackMacs.Text     [ OPTIONAL ]
.INCLUDE TlAsm/FSEqu.Text        [ OPTIONAL ]

[ Here there will be a .PROC or .FUNC directive for each routine, ]
[ followed by the routine itself.  Two examples follow.  ]

; PROCEDURE MyRoutine (count: INTEGER);

.PROC MyRoutine

MyRoutine
      [ the code of MyRoutine ]

; FUNCTION MyOtherRoutine : LongInt;

.FUNC MyOtherRoutine

MyOtherRoutine
      [ the code of MyOtherRoutine ]

.END
```

Note: The .PROC or .FUNC directive clears the symbol table, so symbols defined in one routine can't be referred to in another (without an explicit reference using .REF). If you want to share code between routines, you can instead have a single .PROC directive for SampA followed by a .DEF directive for each routine name.

Including unneeded files with .INCLUDE directives will do no harm except make your program take longer to assemble. The files marked as optional above are the least commonly needed; even some of the others may not be required. Here's what the files contain:

File name	Description
TlAsm/SysEqu.Text	System equates
TlAsm/SysTraps.Text	System traps
TlAsm/SysErr.Text	System error equates
TlAsm/QuickEqu.Text	QuickDraw equates
TlAsm/QuickTraps.Text	QuickDraw traps
TlAsm/ToolTraps.Text	Toolbox traps, except QuickDraw
TlAsm/ToolEqu.Text	Toolbox equates, except QuickDraw
TlAsm/PrEqu.Text	Equates for Printing Manager
TlAsm/SANEMacs.Text	Macros and equates for Floating-Point Arithmetic and Transcendental Functions Packages
TlAsm/PackMacs.Text	Macros and equates for other packages
TlAsm/FSEqu.Text	File system equates

If you've created any similar files for units of your own, just add .INCLUDE directives for them after the last .INCLUDE directive shown above.

To specify the beginning of a segment in assembly language, you can use the directive

```
  .SEG 'segname'
```

where segname is the segment name, a sequence of up to eight characters.

For each assembly-language routine invoked from Pascal, the Pascal source file for your application will include an external declaration. For example:

```
  PROCEDURE MyRoutine (count: INTEGER); EXTERNAL;
  FUNCTION  MyOtherRoutine : LongInt; EXTERNAL;
```

If the routines form a unit that may be used by other applications, you should instead prepare a Pascal interface file for the unit and include it in the USES clause in the application's source file.

You'll assemble the Work/SampA.Text file as shown below.

Prompt	Response
Workshop command line	A [for Assemble]
Input file - [.TEXT]	Work/SampA <ret>
Listing file (<CR> for none) - [.TEXT]	<ret> [for none]
Output file - [Work/SampA] [.OBJ]	<ret> [for Work/SampA.Obj]

Note: If you do want a listing file, you may want to put a .NOLIST directive before your first .INCLUDE and a .LIST after your last one, so the contents of all the included files won't appear in the listing.

You can assemble the code manually and then, after you've created or changed the Pascal source file, use the exec file for the application as illustrated earlier (adding the name of the assembly-language object file to the list of Linker input files). You may also want to set up an exec file that just assembles the assembly-language routines and links the resulting object file with everything else, for when you've changed only those routines and not the Pascal program. This exec file would begin with the responses listed above and then continue with step 2 of the exec file illustrated earlier.

If the entire application is written in assembly language, the source file will have the same structure as the one shown above, but at the beginning of the main program you'll have a .MAIN directive:

```
 .MAIN SampA
```

Even if you have nothing to link your program with, link it by itself; the Linker will put it into a format that RMaker can accept.

SUMMARY OF PUTTING TOGETHER AN APPLICATION

This summary assumes the file-naming conventions presented in the "Getting Started" section. Page numbers indicate where details may be found.

ONE TIME ONLY:

-Prepare a Macintosh system disk by copying the System Folder from the Mac Build disk to a new Macintosh disk (page 4).

-On the Lisa, use the Editor (via the Edit command) to create the exec file (page 14).

ONCE PER VERSION OF YOUR APPLICATION'S SOURCE/RESOURCES:

-On the Lisa, use the Editor to create or edit the application source file (page 6) or the Resource Compiler input file for your application's resources (page 7).

-Insert the Macintosh system disk into the Lisa.

-On the Lisa, run the exec file (page 17). It will eject the Macintosh disk when done.

-To try out your application, remove the disk from the Lisa, insert it into the Macintosh, and open the application's icon.

-When appropriate, back up your working files by using the Backup command in the File Manager to copy Work/= to Back/=, or onto a 3~1/2-inch disk (with, for example, Backup Work/= to -lower-=).

Note: If you create an icon for your application (or modify it), you must delete the invisible desktop file (page 17).

Lisa Workshop Update

The Workshop 3.9 Update is an update to the Pascal Workshop 3.0 release. It contains the latest, most up-to-date Workshop tools, including a new "Post-3.0" Pascal compiler with supporting libraries, the latest SANE libraries, and new versions of the Workshop Shell, the linker, the editor, the assembler, RMaker, and MacCom. The only Workshop 3.0 users who should *not* install the entire update are those who are still using the "Old World" SANE floating point described in the February Supplement.

The Workshop 3.9 Update consists of two disks, **Lisa Workshop Update 1** and **Lisa Workshop Update 2.** These disks have been formatted using the Lisa Pascal Workshop version 3.0, so *they cannot be read from a Lisa running Workshop 2.0 or from a Macintosh.* However, there is nothing on these disks which is useful to anyone who does not have Pascal Workshop 3.0. The disks contain the following files:

Files on Lisa Workshop 3.0 disk **Lisa Workshop Update 1**:
```
Assembler.obj
Code.obj
Editor.obj
IOSPasLib.obj
IUManager.obj
Linker.obj
Mac.boot
MacCom.obj
StartUpdate.text
tmp/ContinueOrAbort.text
tmp/DoUpdate.text
tmp/GetDisk.text
tmp/YesNoFunc.text
```

Files on Lisa Workshop 3.0 disk **Lisa Workshop Update 2**:
```
intrfc/SANELib.text
obj/SANELib.obj
obj/SANELibAsm.obj
OSErrs.Err
Pascal.obj
PasErrs.Err
RMaker.obj
Shell.Workshop
```

How to install the Workshop 3.9 Update:

An automatic exec update procedure is provided to facilitate the installation of the new software. It should work on all configurations of Workshop 3.0, even if Lisa 7/7 is installed on the same disk. Before starting, make sure that you have at least 1000 blocks free on the hard disk that you will be updating (this is recommended for any work with the Workshop). If necessary, you may be able to free up some space by booting from another Workshop profile and Scavenging or booting from **Pascal 3.0 disk 1** and choosing "*Repair*"). Insert the **Lisa Workshop Update 1**disk (note that the Workshop will not accept write-protected disks) and then invoke the "`StartUpdate.text`" exec file by using the run command as follows:

```
R<-lower-StartUpdate
```

The update exec files will lead you through the rest of the update procedure (including allowing you to choose which hard disk to update, prompting you to install **Lisa Workshop Update 2** disk when necessary, deleting the `tmp/` files it uses, and finally asking you to reboot).

The Writeln Window

PasLib (Versions 0.6 and later) allows programmers to capture all Writeln output and handle it in any convenient way. Using this capability, we have written WritelnWindow, a Pascal unit that captures writelns and displays them in a regular window. This unit is intended for **DEBUGGING PURPOSES ONLY**. DO NOT USE IT IN A PRODUCT FOR RELEASE.

Features

- Automatically saves the last N lines of output. N can be any number subject to memory limitations.
- The unit handles all events directed to the output window, including update, activate, and mouse down events. The unit also handles resizing the window and scrolling back through the output.
- Requires .5K of initialization code and 2K of resident code.
- Can be used with any standard Macintosh program.

Release Information

The 5/85 Workshop Supplement 1 disk contains he interface to the unit in the files intrfc/WritelnWindow.text (human readable) and obj/WritelnWindow.obj (machine readable). The source to the unit is in intrfc/WritelnWindow2.text on the 5/85 Workshop Supplement 2 disk.

To use this unit, you must hook it into your Lisa Pascal application in a number of places. NOTE: You must use V.0.7 of Paslib. (PasLib V.0.7 consists of several files; it is included on the 5/85 Workshop Supplement 1 disk). To use the unit, you should include the following lines
in your USES statements:

```
{$U obj/PasLibIntf        }  PasLibIntf,
{$U obj/WritelnWindow     }  WritelnWindow;
```

At the start of your application, call **WWInit**.

```
PROCEDURE WWInit (numLines, numCharsPerLine: INTEGER);
```

After you have initialized the Toolbox, call **WWNew**. Pass this procedure the bounds for the window, its title, whether it should have a goAway box and be visible, and the font and font size to use for output. **WWNew** will allocate a window (in global storage) and setup MacPasLib to send Writeln output to the window.

```
PROCEDURE WWNew (bounds: Rect; windowTitle: Str255; goAway: BOOLEAN;
                 visible: BOOLEAN; linesToSave, outputFont, outputSize:
                 INTEGER);
```

The unit contains five other procedures; they must be called from your event loop. In each case, you must determine if the event is directed to the output window. The global variable **gDebugWindowPtr** contains the WindowPtr for the output window. Test the contents of this variable against the window receiving the event.

The four kinds of events are:

1. **Activate Events**: call **WWActivate** and pass in the modifiers field of the event record.

   ```
   PROCEDURE WWActiveateEvent (modifiers: INTEGER);
   ```

2. **Update Events**: call **WWUpdateEvent**.

   ```
   PROCEDURE WWUpdateEvent;
   ```

3. **Mouse Down Events**: call **WWMouseDown** and pass in the value returned by FindWindow, the mouse point (from the event record) and the modifiers (also from the event record).

   ```
   PROCEDURE WWMouseDown  (where: INTEGER;
                            pt: Point;
                            modifiers: INTEGER);
   ```

4. **Key Down Events:** call **WWReadChr** or **WWReadLn** to capture characters in your window.

   ```
   FUNCTION WWReadChr: char;

   PROCEDURE WWReadLn (Var s:str255);
   ```

The above is the minimum amount of code you need to use this unit in your program. You might want to do other things; for example, if your window has a goAway box, the unit will automatically hide the window if the user clicks in it. Your program would then need to provide a way for the user to make the window visible again. (Call **ShowWindow**, passing it the global variable **gDebugWindowPtr**.) If you want to handle your own scrolling, you can call WWScroll. If you want to handle sizing the window, you also have WWInvalGrowBox and WWGrown.

The file example/DebugWindow.text on the 5/85 Examples 1 disk is an example application which uses the WritelnWindow to display debugging information.

Equate and Glue Files for Lisa Workshop and MDS

The following describes the organization, changes, and new features of the latest release of the MDS and Lisa Workshop interface files. For specific details about the 128K ROM and Hierarchical File System additions, you should refer to **Inside Macintosh Vol. IV**, the draft of which is included in section 18. Also, you should refer to the the latest release of the Tech Notes for futher descriptive information and/or "helpful hints" about using the new features.

NOTE: It is very important that you update your development system with all the latest versions of the interface files. In addition, some routines are present only in newer Systems so you should use the newest System Files available. The interface files for Dec2Str and Str2Dec provided in this supplement depend on package number 7, which is present in the 128K ROM, and in System files 3.2 (and later), but not in earlier System files.

Workshop Pascal Interfaces

A new set of the interface files needed for Lisa Pascal development for the Macintosh is included in this release, which is a beta version of release 2.0. These files contain the additional 128K ROM equates including the equates, traps, and data structures needed to use the new Hierarchical File System. These files should be the basis for all future Macintosh development in Pascal.

Text Files

These files are the interface portions of the various libraries and include the relevant constants, types, and routine definitions. New versions are marked with an *.

`*intrfc/AppleTalk.text`	AppleTalk Pascal interface
`*intrfc/FixMath.text`	Fixed point math
`*intrfc/Graf3D.text`	Three-dimensional graphics routines layered on top of QuickDraw. Use with FixMath.
`intrfc/MacPrint.text`	Device independent printing
`intrfc/MemTypes.text`	Common types
`*intrfc/OSIntf.text`	Operating system routines (Memory Mgr, File Mgr, Sound Driver, RAM serial driver, ...)
`*intrfc/PackIntf.text`	Packages (Standard File, International, Binary-Decimal conversion, Disk initialization, List Manager, ...)
`intrfc/PasLibIntf.text`	PasLib (non built-in) functions dealing with the heap and Writeln redirection.
`*intrfc/QuickDraw.text`	Graphics routines
`*intrfc/SANELib.text`	Standard Apple Numerics Environment (IEEE floating point).
`intrfc/SpeechIntf.text`	MacinTalk (speech synthesis)

`*intrfc/ToolIntf.text`	ToolBox routines (Menu Mgr, Dialog Mgr, Window Mgr, ...)
`*intrfc/SCSIIntf.text`	The interface to the SCSI port manager
`intrfc/WritelnWindow.text`	Debugging window (not for use in products) See issue 2 (Dec. 85 Supplement) for details.
`intrfc/WritelnWindow2.text`	Source to debugging window unit

Object Files

These files are either for use by the Pascal compiler (indicated by $USE), in which case they include the interface definition inside the object file, or for use by the linker (indicated by LINK), in which case they include the actual code to implement the interface, or for both. New files are marked with an *.

`*obj/ABPasCalls.obj`	AppleTalk implementation. LINK with this.
`obj/AppleTalk.obj`	AppleTalk definition. $USE only.
`*obj/FixMathAsm.obj`	Fixed point Math implementation (in assembler). Required for Graf3D. LINK with this.
`*obj/FixMath.obj`	Fixed point Math definition. Required for Graf3D. $USE only.
`*obj/Graf3D.obj`	Definition for fixed point implementation of Graf3D (requires FixMath, does not require SANE). $USE only.
`*obj/Graf3DAsm.obj`	Fixed point implementation of Graf3D (written in assembler). LINK with this.
`obj/MacPrint.obj`	MacPrint definition. $USE only.
`obj/MemTypes.obj`	MemTypes definition. $USE only.
`*obj/OSIntf.obj`	OSIntf definition. $USE only.
`*obj/OSTraps.obj`	OSIntf implementation. LINK with this.
`*obj/PackIntf.obj`	PackIntf definition. $USE only.
`*obj/PackTraps.obj`	PackIntf implementation. LINK with this.
`obj/PasInit.obj`	PasLib initialization implementation of %_BEGIN, %_END and %_TERM. LINK with this.
`obj/PasLib.obj`	PasLib implementation portion in Pascal. LINK with this.
`obj/PasLibAsm.obj`	PasLib implementation portion in assembler. LINK with this.
`obj/PasLibIntf.obj`	PasLib definition. $USE only (if directly calling PasLib routines).

`*obj/PrintCalls.obj`	MacPrint high-level implementation. LINK with this.
`*obj/QuickDraw.obj`	Quickdraw. $USE and LINK.
`obj/RTLib.obj`	PasLib Run Time support--implementation of console I/O. LINK with this.
`*obj/SANELib.obj`	SANE and Elems definition. $USE only. The routine Dec2Str will only work with System files from 3.2 on.
`*obj/SANELibAsm.obj`	SANE and Elems implementation. LINK with this. The routine Dec2Str will only work with Systems from 3.2 on.
`*obj/SCSIIntf.obj`	SCSI port manager interface. $USE only
`obj/SpeechAsm.obj`	MacinTalk (speech synthesis) implementation (written in assembler). LINK with this.
`obj/SpeechIntf.obj`	MacinTalk (speech synthesis) definition. $USE only.
`*obj/ToolIntf.obj`	ToolIntf definition. $USE only.
`*obj/ToolTraps.obj`	ToolIntf implementation. LINK with this.
`obj/WritelnWindow.obj`	Debugging window (not for use in products). $USE and LINK.

Assembler Equates

This release contains new versions of the equate and macro files needed for assembly language development for the Macintosh. This release is a beta version of release 2.0. These files contain the additions required for the 128K ROM and HFS. The files are provided in both Lisa format (TLAsm files) and Macintosh format (for MDS, Macintosh 68000 Development System). These files should be the basis for all future Macintosh assembly language development. New files are marked with an *.

The equates and macros are commented somewhat within the files themselves. More detailed documentation can be found in the appropriate sections of *Inside Macintosh.*

Equate Files

Lisa Workshop:	MDS Files	Contents
`*TLAsm/ATalkEqu.text`	`*ATalkEqu.Txt`	AppleTalk equates and globals
`*TLAsm/Fixmath.text`	`*FixMath.Txt`	Fix-point math equates and globals (see FixMath and Graf3D section)
`*TLAsm/FSPrivate.text`	`*FSPrivate.Txt`	Additional file system equates and globals for debugging use only.

*TLAsm/FSEqu.text	*FSEqu.Txt	File system equates and globals
*TLAsm/Graf3D.text	*Graf3D.Txt	Graf3D (3-D graphics) equates and globals (see FixMath and Graf3D section)
*TLAsm/HardwareEqu.text	*HardwareEqu.Txt	Hardware equates and globals (for debugging use only)
---	MacDefs.Txt	Macros translating Lisa Workshop assembler directives into MDS directives
---	MacTraps.Asm	Creates MacTraps.Sym (MDS symbol file)
*TLAsm/PackMacs.text	*PackMacs.Txt	Package macros
*TLAsm/PrEqu.text	*PrEqu.Txt	Printing equates and globals
*TLAsm/Private.text	*Private.Txt	Additional equates and globals (for debugging use only)
*TLAsm/QuickEqu.text	*QuickEqu.Txt	QuickDraw equates and globals
*TLAsm/SANEMacs.text	*SANEMacs.Txt	Numerics macros (see SANE section)
*TLAsm/SONYEqu.text	*SONYEqu.Txt	Disk driver equates and globals (for debugging use only)
*TLAsm/SysEqu.text	*SysEqu.Txt	Low-level system equates and globals
*TLAsm/SysErr.text	*SysErr.Txt	System error numbers
*TLAsm/ToolEqu.text	*ToolEqu.Txt	Toolbox equates and globals
*TLAsm/Traps.text	*Traps.Txt	All of the trap definitions

The files `SysEqu`, `ToolEqu`, `SysErr`, and `QuickEqu` start with an equate such as "wholeSystem" which is used for conditional assembly. If you do not need the less common equates after ".IF wholeSystem" you can change wholeSystem to 0 and reduce the time and space required for your assembly.

THE NEW FEATURE FLAGS

In a previous release of the above files, two flags were defined, ROM128K and HFSUsed, that were used to "prevent" inadvertent use of the new features. These have been removed from the interface files. If your program defines them, these definitions have no effect (except to take up symbol table space.)

PRIVATE FILES

In previous releases, a distinction was made between "private" and "public" equate files. A "private" file contained information that was either machine specific and/or reserved for internal use by Apple Computer, Inc. Occasionally, these "private" files were given to developers who demonstrated a need for this information.

We have decided to no longer maintain "private" files as distinct from "public" files. From now on, the formerly private files are now part of the general release. However, in order to prevent developers from inadvertently using the contents of these "dangerous" files, a comment has been placed at the head of the file warning the user to beware. Following the comment is an equate which effectively turns off the contents of the file through a conditional statement located immediately following the equate. In order to use the file, the user must consciously set the equate to 1 to turn on the conditional assembly. The following is an example of this, taken from the file `FSPrivate`:

```
; In order to prevent any "accidental" use of this information, it has
; been disabled using the conditional-assembly variable defined below.
; If you change this to a non-zero value, you're on your own.
```

The affected files are:

`FSPrivate` with the flag FSNonPortable
`HardwareEqu` with the flag HWNonPortable
`Private` with the flag PrNonPortable
`SONYEqu` with the flag SONYNonPortable

On a case by case basis, an analysis of the equate files was made that determined whether or not certain equates were public (that is, in *Inside Macintosh* or required by outside developers) or private (that is, only for the use of Apple). In some cases, some previously available equates have been made private. If your program does not assemble with the new equates due to undefined labels, they have more than likely become private. If you are concerned about your application's "private" requirements, contact Technical Support.

TRAP FILES

The original three trap files: `SysTraps, ToolTraps, and QuickTraps` have been merged into one file, `Traps`. **You will have to update your list of include files in your source to use this new file and to remove references to the old versions.**

USING THE NEW FILES

You should update your development system with **all** the new files, not just the ones that have changed. There are incompatibilities between files from different releases, so it is best to use all the files from the latest release.

You may find that your program does not compile/assemble with the newest interface files. This is probably due to one or more of the following reasons:

1. Some equates have been moved either from a "public" file to a "private" file. You may need to include one of the "private" files.

2. If you are using a "private" file, you must modify the equate at the beginning of the file in order to actually use the equates contained therein. (See above.)

3. Throughout the process of setting up these files and with the writing of the latest version of *Inside Macintosh* that describes the new additions, some names have been changed to better reflect the meaning associated with the particular value. If you have or are using one of the interim releases of these files, you may have to change a small number of names used in your program. Sorry!

Note: Some of the routines in these new files require newer System Files. At present, Dec2Str and Str2Dec will only work with version 3.2 or greater system files.

THE NEW PRINTING INTERFACE

In previous releases, there were two files to support printing on the Mac, `PrLink` and `PrScreen`. Both of these files have been replaced by the file `PrintCalls`. This file supports the functionality of both the previous files as well as fixing a few bugs.

FIXMATH AND GRAF3D

Fixmath has previously been available as a library file with which you linked your program. The 128K ROMs now include a superset of the previous version of Fixmath. In order to accommodate programs that will be run on either 64K or 128K ROM systems, the Fixmath library file has been modified to check for 128K ROMs. This code checks which ROMs are available and if the 128K ROMs are present, it will call the appropriate trap; if not, it will do the work. This is only for the calls common between the original version of Fixmath and the new version available with the new ROMs. The new calls are only available with the 128K ROMs as traps and they are defined in the Fixmath interface files. If you want to run only on 128K ROMs and you wish to use Fixmath, you can find the required trap definitions in the Fixmath interface file, where they are included as comments. You can edit this file to make these trap definitions available and remove the references to the externally provided routines. Remember, however, that it is your responsibility to check to see if the 128K ROMs are present!

Graf3D uses Fixmath. Its library file has been modified to call the Fixmath trap if the 128K ROMs are present. To use Graf3D you must link to both the Graf3D and the Fixmath library files.

The Graf3D interface for Pascal has also been modified. *You must now define the pointer to the Graf3D port in your source file.* The first variable defined in your program should be:

```
VAR     thePort3D: Port3DPtr;
```

This change was done in order to simplify the interface for 'C' programs. Also, numerous bugs in Graf3D have been fixed.

SDOpen

Remember that the SERD resource (included in this Supplement) must be pasted into your application in order for the serial driver calls to find it. Almost any resource mover/editor (such as ResEdit) can be used to paste the resource.

MACINTOSH INTERFACE FILE CHANGE HISTORY

The following is a partial change history of the interface files used for the Lisa Workshop and MDS. The files included with this Supplement are the "Beta Release" referred to below. The file `Yanked.txt` contains all of the routines, equates, or other text that has been removed from the interface files.

Assembly Equate Change History

General Changes:

May 85 —> Dec 85 Supplement
ROM128K and HFSUsed Flags added as conditional assembly for all new ROM and HFS features

Dec 85 Supplement —> Beta Release

ROM128K and HFSUsed Flags removed as conditional assembly for all new ROM and HFS features various equates have been moved from file to file, in some cases made private or made public some equates have been removed from the equate files, the following is a list:

from SysEqu

ToolDisp	EQU	10
OldDisp	EQU	9
NewOSTrap	EQU	$200
NewToolTrap	EQU	$600

from Syserr

FSDSIntErr	EQU	-127
memROZWarn	EQU	-99

from ToolEqu

Resorse	EQU	$5
gPortSize	EQU	108
TopMenuItem	EQU	$B26
AtMenuBottom	EQU	$B28
MicroSoft	EQU	ApplScratch

from Private.a

tagMask	EQU	$C0000000
bcOffMask	EQU	$0F000000
bcMask	EQU	$00FFFFFF
ptrMask	EQU	$00FFFFFF
handleMask	EQU	$00FFFFFF
HeapStart	EQU	$1400
DskWr11	EQU	$12F
SoundLast	EQU	$282
Filler3A	EQU	$214
BasicGlob	EQU	$2B6
endofvars	EQU	$340
ToolVars	EQU	$980
GrafBegin	EQU	$800

```
GrafVar          EQU    $824
GrafEnd          EQU    $8F2
lGlobals         EQU       0
LastTGLobal      EQU    $AFC
ToolGBase        EQU    $980
Checking         EQU       0
Statistics       EQU       0
Robust           EQU       0
CountMPs         EQU       0
DfltFlags        EQU       0
FSDSErr          EQU     -59
KensOK           EQU       1
```

<u>from HardwareEqu</u>
all onMidMac values removed

ATALKEQU

<u>May 85 —> Dec 85 Supplement</u>
capitalization changes to reflect InsideMac

<u>Dec 85 Supplement —> Beta Release</u>
error equates moved to SYSERR

FIXMATH

<u>Dec 85 Supplement —> Beta Release</u>
new file, 128K ROM traps added

FSEQU

<u>May 85 —> Dec 85 Supplement</u>
a number of equates have been moved to SYSEQU and FSPRIVATE
HFS equates added

<u>Dec 85 Supplement —> Beta Release</u>
references to TFS changed to HFS
directory master block, file entry, working directory, and PMSP equates added

FSPRIVATE

<u>Dec 85 Supplement —> Beta Release</u>
file made public

GRAF3D

no change

HARDWAREEQU

<u>May 85 —> Dec 85 Supplement</u>
VIA, SCC, and disk address equates moved to SYSEQU

<u>Dec 85 Supplement —> Beta Release</u>
directory master block, file entry equates moved to FSEQU
all equates for "onMidMac" removed

PACKMACS

<u>May 85 —> Dec 85 Supplement</u>
list manager equates added

Dec 85 Supplement —> Beta Release
datahandle changed to userhandle for list manager

PREQU

May 85 —> Dec 85 Supplement
PrintErr and ChooserBits moved to PRIVATE

Dec 85 Supplement —> Beta Release
PrintErr added as comment only

PRIVATE

Dec 85 Supplement —> Beta Release
file made public

The following have been placed back into this file:

```
FOutError           EQU    $998
FOutFontHandle      EQU    $99A
FOutBold            EQU    $99E
FOutItalic          EQU    $99F
FOutULOffset        EQU    $9A0
FOutULShadow        EQU    $9A1
FOutULThick         EQU    $9A2
FOutShadow          EQU    $9A3
FOutExtra           EQU    $9A4
FOutAscent          EQU    $9A5
FOutDescent         EQU    $9A6
FOutWidMax          EQU    $9A7
FOutLeading         EQU    $9A8
FOutUnused          EQU    $9A9
FOutNumer           EQU    $9AA
FOutDenom           EQU    $9AE
```

QUICKEQU

No changes.

QUICKTRAPS

May 85 —> Dec 85 Supplement
128K ROM routines added

Dec 85 Supplement —> Beta Release
file contents moved to TRAPS

SANEMACS

May 85 —> Dec 85 Supplement
scanner and formatter functions added

Dec 85 Supplement —> Beta Release
bug fix to the FDecStr macro

SCSIEQU

Dec 85 Supplement —> Beta Release
error code names changed

communication error equates removed all references to TFS changed to HFS
command block equates' names changed
HFSID value changed from 'HFS1' back to 'TFS1'

SONYEQU

Dec 85 Supplement —> Beta Release
file made public

SYSEQU

May 85 —> Dec 85 Supplement
hardware equates commented out for VIA, SCC, and IWM
128K ROM and HFS additions

Dec 85 Supplement —> Beta Release
finder and directory info record equates changed from "value+offset" to just "offset"
hardware equates uncommented out for VIA, SCC, and IWM
interrupt and auto vector equates moved to HARDWAREEQU

SYSERR

May 85 —> Dec 85 Supplement
128K ROM and HFS additions

Dec 85 Supplement —> Beta Release
AppleTalk errors added
errors arranged in numerical order

SYSTRAPS

May 85 —> Dec 85 Supplement
HFS and memory manager traps added

Dec 85 Supplement —> Beta Release
file contents moved to TRAPS

TIMEEQU

Dec 85 Supplement - Beta Release
new file

TOOLEQU

May 85 —> Dec 85 Supplement
Taliesin font name changed to Mobile
128K ROM additions

TOOLTRAPS

May 85 —> Dec 85 Supplement
128K ROM traps added

Dec 85 Supplement —> Beta Release
file contents moved to TRAPS

TRAPS

Dec 85 Supplement —> Beta Release
new file made from QUICKTRAPS, SYSTRAPS, and TOOLTRAPS

Pascal Interface Change History

APPLETALK
No Changes

FIXMATH
Dec 85 Supplement —> Beta Release
new 128K ROM traps added, interface restructured to support both RAM and ROM based calls

GRAF3D
Dec 85 Supplement —> Beta Release
new file

MACPRINT
No changes

MEMTYPES
No changes

OSINTF
Dec 85 Supplement —> Beta Release
new HFS calls and data structures added
new 128K ROM calls added

PACKINTF
Dec 85 Supplement —> Beta Release
list manager calls and data structures added

QUICKDRAW
Dec 85 Supplement —> Beta Release
new 128K ROM calls added

SCSIINTF
Dec 85 Supplement —> Beta Release
new file

TOOLINTF
Dec 85 Supplement —> Beta Release
new 128K ROM calls and data structures added
major addition of data structures for Font Mgr

"GLUE" and .Rel File Change History

APPLETALK (ABPASCALLS)
Dec 85 Supplement —> Beta Release
bug fixes

FIXMATHASM
Dec 85 Supplement —> Beta Release
modified so that the RAM calls will call the 128K ROM if the ROM is present

GRAF3DASM

Dec 85 Supplement —> Beta Release
modified so that the code calls the ROM-based Fixmath calls
if the ROM is present
major bug fixes

OSTRAPS

Dec 85 Supplement —> Beta Release
added glue for new HFS and 128K ROM calls
bug fixes

PACKTRAPS

Dec 85 Supplement —> Beta Release
added glue for list manager

PRINTCALLS

Dec 85 Supplement —> Beta Release
bug fixes

QUICKDRAW

No changes

SCSITRAPS

Dec 85 Supplement —> Beta Release
new file - bug fixes since preliminary release

TOOLTRAPS

Dec 85 Supplement —> Beta Release
added new 128K ROM call glue

PPostEvent

Developers have clamored for a version of

```
FUNCTION PostEvent(eventCode: INTEGER; eventMsg: LONGINT): OSErr;
```

that would also return a pointer to the queue element created. Alas, the ROM actually leaves the desired value in A0, but it gets smashed by the OS trap interface which, in this case, is set to save/restore A0 across the _PostEvent trap.

What has been added to `Traps` is:

```
        .TRAP      _PPostEvent    $A12F
```

and to `OSIntf`

```
FUNCTION PPostEvent(eventCode: INTEGER; eventMsg: LONGINT; VAR qEl: EvQEl): OSErr;
```

NGetTrapAddress AND NSetTrapAddress

Since the Tool and OS trap tables have been disentangled and expanded to their fullest extents, the GetTrapAddress and SetTrapAddress routines must likewise be extended. In the 64K ROM, a trap's number in the table implied its Tool/OS status, whereas in the new ROM this must be specified separately, EXCEPT when, for compatibility, the 64K ROM numbering is used. The 64K ROM routines are:

```
FUNCTION  GetTrapAddress(trapNum: INTEGER): LongInt;

PROCEDURE SetTrapAddress(trapAddr: LongInt; trapNum: INTEGER);
```

Along with these, the two new routines (and a relevant enumerated type) are:

```
TYPE TrapType = (OSTrap, ToolTrap);

FUNCTION  NGetTrapAddress(trapNum: INTEGER; tTyp: TrapType): LongInt;

PROCEDURE NSetTrapAddress(trapAddr: LongInt; trapNum: INTEGER; tTyp: TrapType);
```

Internally, these amount to about the same code as the old routines, but with appropriate modifier bits set in the trap word in order to distinguish OS and Tool requests. Two new equates have been added to Traps to accommodate the modifier bits in the GetTrapAddress and SetTrapAddress calls:

```
        newTool    EQU          $0600
        newOS      EQU          $0200
```

```
; File: SuppYanked.txt
;
; Version 1.0a2
;
; Copyright 1984, 1985, 1986 Apple Computer, Inc. All Rights Reserved
;

; This is a private file. This is a special version of Yanked.txt done
; for the Software Supplement (Volume I Issue 3) (6/27/86)

;The following items have been removed from the standard
;set of include files.

;---------------------------------------------------------------
;
; The following information was formerly in "private" files that were
; not released to the general developer community.
;
; The information in this file is not needed for normal application
; development. These equates and macros were necessary for development
; of the Macintosh ToolBox and Operating System, and are likely to be
; dependent on their current implementation. Use of any information
; in this file is likely to cause your software to fail on future
; versions of Macintosh system software or hardware.


;                    WARNING!!!!!!!!!!
; Many of the equates in this "Yanked" file are not valid for all systems.
; This information has been provided so as to make development easier
; should you find that your code references something that has been
; removed from the standard, non-private equate files.
;
; Apple Developer Support will not support any use of the following
; information.
;
; In order to prevent any "accidental" use of this information, it has
; been disabled using the conditional-assembly variable defined below.
; If you change this to a non-zero value, you're on your own.

YankedNonPortable   EQU 0   ;exclude the NonPortable contents of this file

    IF YankedNonPortable THEN

;from SysEqu

ToolDisp     EQU    10              ; bit #10 distinguishes Tool = 1 or OS = 0 Get/Set Trap address
OldDisp      EQU    9               ; bit #9 distinguishes Old = 0 or New = 1 trap numbering
NewOSTrap    EQU    $200            ; for Get/SetTrap Address
NewToolTrap  EQU    $600            ; for Get/SetTrap Address

;from Syserr

FSDSIntErr  EQU    -127         ; Internal file system error
memROZWarn  EQU    -99          ; soft error in ROZ

;from ToolEqu
```

```
Resorse       EQU   $5             ; bit test = Resorse  --- for RMGR
gPortSize     EQU  108             ; a grafPort is 108 bytes
TopMenuItem   EQU  $B26            ; (word) used for menu scrolling
AtMenuBottom   EQU  $B28           ; (word) flag for menu scrolling
MicroSoft     EQU  ApplScratch     ; for Seattle [12 Bytes]

; from Private.a

FOutError        EQU  $998     ; error code
FOutFontHandle   EQU  $99A     ; handle to font bits
FOutBold         EQU  $99E     ; bolding factor
FOutItalic       EQU  $99F     ; italic factor
FOutULOffset     EQU  $9A0     ; underline offset
FOutULShadow     EQU  $9A1     ; underline halo
FOutULThick      EQU  $9A2     ; underline thickness
FOutShadow       EQU  $9A3     ; shadow factor
FOutExtra        EQU  $9A4     ; extra horizontal width
FOutAscent       EQU  $9A5     ; height above baseline
FOutDescent      EQU  $9A6     ; height below baseline
FOutWidMax       EQU  $9A7     ; maximum width of character
FOutLeading      EQU  $9A8     ; space between lines
FOutUnused       EQU  $9A9     ; unused byte (must have even number)
FOutNumer        EQU  $9AA     ; point for numerators of scale factor
FOutDenom        EQU  $9AE     ; point for denominators of scale factor

tagMask          EQU  $C0000000 ; Mask for the 2-bit Tag Field
bcOffMask        EQU  $0F000000 ; Mask for the 4 bit Byte Count offset
bcMask           EQU  $00FFFFFF ; Mask for the 24 bit Byte Count
ptrMask          EQU  $00FFFFFF ; Mask pointer to low 24 bits
handleMask       EQU  $00FFFFFF ; Mask handle to low 24 bits
HeapStart        EQU  $1400     ; on Mac+ only
DskWr11          EQU  $12F     ; try 1-1 disk writes? [byte]
SoundLast        EQU  $282     ; address past last sound variable
Filler3A         EQU  $214     ; used by standard file
BasicGlob        EQU  $2B6     ; Basic globals [pointer]
endofvars        EQU  $340     ; end of final defined vars
ToolVars         EQU  $980     ; toolbox variables
GrafBegin        EQU  $800     ; graf global area
GrafVar          EQU  $824
GrafEnd          EQU  $8F2     ; end of graphics globals

; Miscellaneous Constants

lGlobals         EQU  0    ; quickdraw globals accessed 0(A5)

; Miscellaneous Globals

LastTGLobal      EQU  $AFC     ; address of last global
ToolGBase        EQU  $980     ; base address of toolbox globals

;_______________________________________________________________________
;
; These equates come from the old equate file HeapDefs.
;    These equates are private to Apple Computer, Inc. and should be used
;    solely for building system software.
;
;_______________________________________________________________________
```

```
Checking        EQU    0            ;check arguments and data structures
Statistics      EQU    0            ;gather statistics on usage
Robust          EQU    0            ;enables super-robust internal checks
CountMPs        EQU    0            ;enables counting of master pointers

DfltFlags       EQU    0            ;Checking is on when zone is init'd


;______________________________________________________________________
;
; This equate comes from the old equate file SysErr.a
;    This equate is private to Apple Computer, Inc. and should be used
;    solely for building system software.
;
;______________________________________________________________________


FSDSErr     EQU    -59      ; file system deep s--t error:
                            ;  during rename the old entry was deleted but could
                            ;   not be restored . . .

KensOK      EQU    1                ; set to 0 to make it work


; from HardwareEqu

HiIntMask       EQU    $0700           ; mask for all interrupts
SCCIntMask      EQU    $0600           ; SCC interrupt vector
VIAIntMask      EQU    $0400           ; VIA interrupt vector
DMAIntMask      EQU    $0200           ; DMA interrupt vector
LoIntMask       EQU    $0100
SCCEnblMask     EQU    $FDFF           ; mask to enable SCC interrupts


ScreenLow       EQU  $610000        ; top of screen screen address for vid page 1
ROMStart        EQU  $400000        ; starting address of ROM code
ROMBHi          EQU  $40            ; high byte of ROM address
SoundLow        EQU  $0FFD00        ; sound buffer start address
PWMBuffer       EQU  $0FFD01        ; PWM bytes are low bytes
MaXRAM          EQU  $100000        ; last RAM address + 1

RAMMAP          EQU  $800000        ; base address of MMU

SERegs          EQU    ScreenLow + $A000   ; Sys Error Regs w/o Overlay (uses part of spare video mem
SEOvlyRegs      EQU    SERegs            ; no overlay space

SEScrnNum       EQU    ScreenLow + $50C7   ; Sys Error Number Screen address = $4628+80*34 - 1
                                            ; "vaguely centered" plus 34 scanlines -1 for extra digit
SEScrnInc       EQU    ScreenLow + $48AA   ; address to diddle below death number=$4628+80*8+2
SEScrnIcon      EQU    ScreenLow + $4628   ; centered screen addr
SEScrnFace      EQU    ScreenLow + $4809   ; screen addr for face = $4628+80*6+1
SEScrnI2        EQU    ScreenLow + $4AD9   ; screen addr for boot icon overlay=$4628+80*15+1


DSrectTL        EQU    $0078003C        ; top left = 120,60
DSrectBR        EQU    $01180244        ; bottom right = 280,580
CtrDiff         EQU    $00450040        ; coord diff from Mac screen center
```

```
scrnRowB        EQU    80
maxX            EQU    640
maxY            EQU    480
scrnBytes       EQU    38400         ; scrnRowB*maxY


VBase           EQU  $E80000    ; base address
AVBufB          EQU  $E80000    ; buffer B
AVBufA          EQU  $E81E00    ; buffer A
AVBufM          EQU  $E80000    ; buffer with mouse button bit
AVIFR           EQU  $E81A00    ; interrupt flag register
AVIER           EQU  $E81C00    ; interrupt enable register

; IFR bits:

ifCA2           EQU    0        ; CA2, ONESEC interrupt
ifCA1           EQU    1        ; CA1, *CMDCOMP signal
ifSR            EQU    2        ; SR done, Serial i/o to Servo
ifCB2           EQU    3        ; CB2, Serial Data from Servo
ifCB1           EQU    4        ; CB1, (unused as interrupt)
ifT2            EQU    5        ; T2, INDEX pulse counter (VBL ctr)
ifT1            EQU    6        ; T1, (unused as interrupt)
ifIRQ           EQU    7        ; any interrupt


TicksPr100      EQU    157      ; timer 1 setting for 100 usec intervals
OneSecConst     EQU    4        ; gets converted to $40000 for onesec constant

SCCRBase        EQU  $C80001    ; SCC base read address
SCCWBase        EQU  $C80001    ; SCC base write address

sccWrite        EQU  $0     ; general offset for write from read

DBase           EQU  $D80001    ; disk address base
DPh0L           EQU  $D80001    ; phase 0 low
DPh0H           EQU  $D80201    ; phase 0 high
DMtrOff         EQU  $D81001    ; IWM Motor off
DMtrOn          EQU  $D81201    ; IWM Motor on
DiskQ6L         EQU  $D81801    ; shift register
DiskQ6H         EQU  $D81A01
DiskQ7L         EQU  $D81C01
DiskQ7H         EQU  $D81E01

; DMA (68450) addresses, offsets

DMABase         EQU  $E00000    ; base address
dCSR            EQU  $00        ; Channel Status Reg (byte)
dCER            EQU  $01        ; Channel Error Reg (byte)
dDCR            EQU  $04        ; Device Control Reg (byte)
dOCR            EQU  $05        ; Operation Control Reg (byte)
dSCR            EQU  $06        ; Sequence Control Reg (byte)
dCCR            EQU  $07        ; Channel Control Reg (byte)
dNIV            EQU  $25        ; Normal Interrupt Vct (byte)
dEIV            EQU  $27        ; Error Interrupt Vct (byte)
dCPR            EQU  $2D        ; Channel Priority Reg (byte)
dMFC            EQU  $29        ; Memory Function Codes (byte)
dDFC            EQU  $31        ; Device Function Codes (byte)
```

```
dBFC            EQU  $39        ; Base Function Codes (byte)
dMTC            EQU  $0A        ; Memory Transfer Ctr (word)
dBTC            EQU  $1A        ; Base Transfer Ctr (word)
dMAR            EQU  $0C        ; Memory Address Reg (long)
dDAR            EQU  $14        ; Device Address Reg (long)
dBAR            EQU  $1C        ; Base Address Reg (long)
dGCR            EQU  $FF        ; General Control Reg (byte)

;   base address for SCSI Port

scsiRd      EQU         $D00001         ; base address of SCSI interface - READ
scsiWr      EQU         $D00001         ; base address of SCSI interface - WRITE
SCSIVct     EQU         AutoInt3    ; SCSI int at level 3

onesec      EQU    191056           ; looptimes
halfsec     EQU    onesec/2
OneSecTicks EQU    68               ; ticks, of course
StlDelay    EQU    $60              ; default bus settle delay

ENDIF  ;end exclusion of private information
```

Resource File Builder (RFB)

Introduction

RFB is a Resource File Builder tool for the Lisa Workshop. A version of RFB for Workshop 3.9 can be found in the file `RFB.obj` on the **Examples 3** disk. The source for the RFB tool (including a Workshop 3.9 exec file) can be found in `source/RFB.text` on the same disk (see that file for more information). Workshop 2.0 users might be able to modify the source to make it work on their system.

In the course of development of "Please" (Hayes' database management package) for the Macintosh, it became apparent that a lot of time could be saved by avoiding RMaker resource compiles as much as possible. Further, as the program grew it was decided that it should be split into separate files, to allow the user to have only those resources on online disks that were necessary to perform the desired functions. None of the programs distributed with the Lisa Pascal Workshop proved sufficient to the task, so Toby Nixon, Software Analyst at Hayes, wrote the Resource File Builder (RFB) program.

RFB runs on the Lisa. It allows the developer to produce a Macintosh resource file on the Lisa from one or more other existing resource files. There are several specific advantages gained by using the program, including:

- Non-CODE resources, such as STR#s, ALRTs, DLOGs, CTRLs,etc., can be placed in an RMaker input file and compiled once. A separate RMaker input file can be prepared which specifies ONLY the CODE resources. After each Link, only the CODE resource file must be reprocessed by RMaker. RFB combines the two resource files to produce the file that is MacCom'ed.

- Resources such as FONTs and PICTs, which are extremely difficult to produce in the Workshop, can be produced on the Mac (or Lisa under MacWorks), MacCom'ed onto the Workshop disk, and combined with other resources with RFB to produce the final Mac resource file.

- RFB allows an application to be easily split into separate resource files, such as a main file, utility file, and help file. The application running on the Mac can open the auxiliary files when necessary (usually because the user initiated a function that requires CODE or other resources from that file) using OpenResFile. The Mac Resource Manager automatically searches all open resources files, so no special code is needed in the application to find the resources once the files are open.

- When several programmers or other personnel (such as technical writers working on help text) are involved in the creation of non-CODE resources, they can separately process their work through RMaker, and distribute their files to the others. Each would run RFB to produce the executable version of the program, without having to wait for RMaker to process all of the other team members' work.

Running RFB

Execute RFB by using the Workshop "R" (run) command. RFB is most often run from an exec file which includes all of the RFB input to produce the desired output. RFB requests the name of the output file to be produced. It does not check to see if the file already exists. It does not currently append any default extension, so the entire file name must be given. If a null entry is made, the program terminates.

RFB then requests the name of the input resource file to read. The entire file name must be entered. If the file is not found, an appropriate error message is displayed. If the file is found, it is opened and the resource map is read into memory. If no entry is made, the output resource file map is written, and the output file is closed. RFB then returns to allow the specification of another output file.

RFB then requests the resource type to be copied. If a null entry is made, RFB closes the input file and requests a new input file name. If "*" is entered, then all resources of all types are copied from the input file to the output file. Otherwise, the type code entered is search for the the input file map, and an appropriate error message is displayed if it is not found. If it is found, an entry is made in the output file map for that type, and processing continues.

If the type specified is CODE, RFB asks if you want to specify code segments by name rather than number. This is very useful if the program is under active development, with frequent addition and deletion of code segments. If YES is specified, RFB then asks for the name of the linkmap file associated with the current input file. The linkmap file is opened and scanned, and the names and associated numbers of all code segments are tabulated.

After the type is entered, RFB requests the resource ID numbers to copy. If a null entry is made, RFB returns to request the next type (if no resources have been copied for the type, the type entry is removed from the output map). If "*" is entered, all resources of the current type are copied to the output file. If CODE segments are being specified by name, the name is translated to the CODE resource ID number (segment number) for copying. If the specified resource is already in the output map, an error message is displayed.

While specifying CODE segments by name, it is still possible to specify by number. When a CODE segment name is requested, simply enter '#' followed by the number desired. This is necessary when requesting the jump table (#0) and the blank segment (#1). Resources of type VERS are handled a special way. When RFB starts, it gets the system date and time. When a VERS resource is copied, the resource data from the input file is replaced by the 10-byte system date-time stamp. All VERS resources written during a single run of RFB will have exactly the same time stamp data. This allows the application at run time to insure that auxiliary files are of the same version as the main program resource file, by simply doing a byte-by-byte comparision of the VERS resources in the files.

RFB Limits
RFB is currently limited by static array bounds to:

25 resource types in the output file.
128 references to any one type.
6144 bytes maximum input resource map size.
256 CODE segments in an input link map.

RFB files included on the **Examples 3** disk:

`RFB.obj`	Object code for RFB
`RFB/exec.text`	The procedure file used to build Please on the Macintosh, included as an example of an RFB exec file.
`source/RFB.text`	Source code for RFB

Questions can be address to Toby Nixon of Hayes Microcomputer Products at (404) 449-8791.