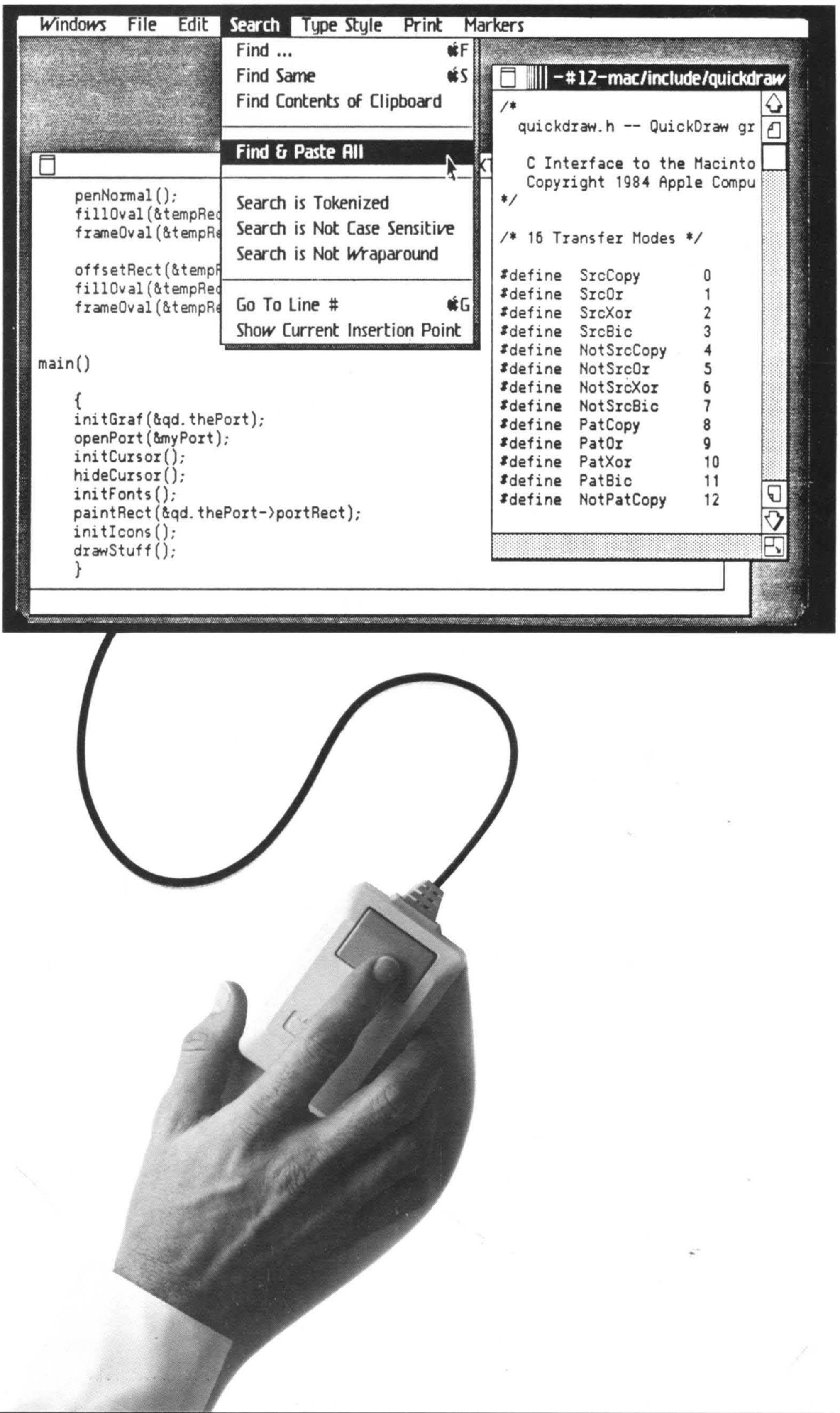
Lisa® Workshop C System


Windows File Edit Search Type Style Print Markers
Find ... ⌘F
Find Same ⌘S
Find Contents of Clipboard
Find & Paste All
Search is Tokenized
Search is Not Case Sensitive
Search is Not Wraparound
Go To Line # ⌘G
Show Current Insertion Point
-#12-mac/include/quickdraw

```
    penNormal();
    fillOval(&tempRec
    frameOval(&tempRe

    offsetRect(&tempR
    fillOval(&tempRec
    frameOval(&tempRe


main()

    {
    initGraf(&qd.thePort);
    openPort(&myPort);
    initCursor();
    hideCursor();
    initFonts();
    paintRect(&qd.thePort->portRect);
    initIcons();
    drawStuff();
    }
```

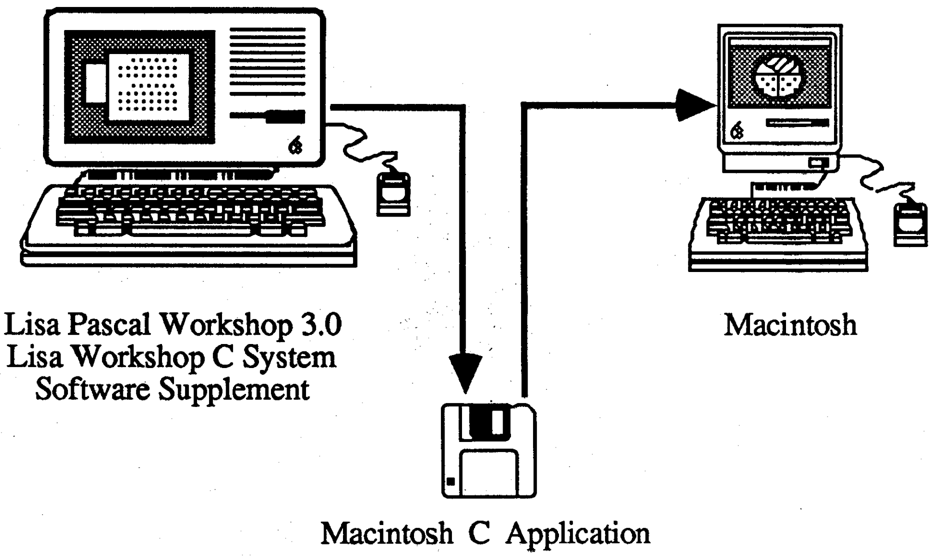
```
/*
  quickdraw.h -- QuickDraw gr

   C Interface to the Macinto
   Copyright 1984 Apple Compu
*/

/* 16 Transfer Modes */

#define  SrcCopy      0
#define  SrcOr        1
#define  SrcXor       2
#define  SrcBic       3
#define  NotSrcCopy   4
#define  NotSrcOr     5
#define  NotSrcXor    6
#define  NotSrcBic    7
#define  PatCopy      8
#define  PatOr        9
#define  PatXor       10
#define  PatBic       11
#define  NotPatCopy   12
```

Lisa Workshop
C User's Guide



Acknowledgments

Apple, Lisa, and the Apple logo are registered trademarks of Apple Computer, Inc.
Macintosh is a trademark of McIntosh Laboratory, Inc. and is being used with express permission of its owner.
UNIX is a trademark of AT&T.
VAX and PDP are trademarks of Digital Equipment Corporation.

Lisa Workshop
C User's Guide

Table of Contents

Preface

This manual contains information you need to write C programs for the Macintosh™ by developing, compiling, and linking your programs under the Lisa® Workshop. Workshop C consists of a C compiler developed by Green Hills Software, the Standard C Library, and the Macintosh Interface Libraries.

Contents of this Manual

This manual contains the following sections:

- Section 1... a definition of the Workshop C Language.
- Section 2... a clarification of implementation-specific details of the language definition.
- Section 3... notes describing calling conventions and other implementation issues.
- Section 4... details on how to run the compiler, including a list of compiler options.
- Section 5... a list of library files and notes on when to link with each file.
- Section 6... a complete definition of the Standard C Library provided with Workshop C.
- Section 7... the C definition of the Macintosh Interface Libraries.
- Section 8... a combined index to the Standard C Library and the Macintosh Interface Libraries.

Reference Materials

You'll need to be familiar with these additional reference materials:

- *Lisa Workshop User's Guide*, Pascal Workshop 3.0, Apple Computer Inc.
- *The C Programming Language*, Kernighan and Ritchie (Prentice-Hall, 1978).
- *Inside Macintosh.*
- *Apple Numerics Manual.*
- "Putting Together A Macintosh Application," Macintosh Software Supplement, Apple Computer Inc.

Inside Macintosh and the *Apple Numerics Manual* will be published by Addison-Wesley in late 1985. A preliminary, promotional edition of *Inside Macintosh* is currently available from Apple Computer. The contents of the *Apple Numerics Manual* are available in the "Standard Apple Numeric Environment" manual in the Lisa Systems Software binder of the Pascal Workshop 3.0 documentation.

Installation Notes

Lisa Workshop C is intended for use with the May 1985 Macintosh Software Supplement, which includes the Lisa Workshop 3.9. The files listed below are on the Workshop C release disks. They may be used directly or copied to a hard disk.

Compiler

compiler	c.obj	Version 1.8.3

Macintosh Sample Program

make file	mac/c/make.text
source	mac/c/sample/qdsample.c.text
resource file	mac/c/sample/qdsample.r.text

Macintosh Libraries

headers	mac/c/include/... .h.text
objects	mac/c/lib/... .obj

Lisa Workshop
C User's Guide

1. The Workshop C Language

Workshop C is a complete implementation of the C programming language. *The C Programming Language* by Kernighan and Ritchie (Prentice-Hall, 1978) is currently the most authoritative written definition of C. However, the language has changed in several ways since the book was written. In addition, numerous details of the language definition are open to interpretation. Therefore, the de facto standard definition of C differs in several ways from the language originally defined by Kernighan and Ritchie. This de facto standard is loosely defined by the Portable C Compiler (PCC), the most widely used implementation of C.

Standard C is our name for the de facto standard definition of C as defined and implemented by the Berkeley 4.2 BSD VAX implementation of PCC, including the documented Western Electric extensions. Workshop C is based on this de facto standard (not on the proposed ANSI standard currently under development).

In addition, Apple has extended the standard language definition in several ways to facilitate writing programs for the Macintosh. Workshop C includes type void, enumeration data types, structure function parameters and results, enumeration data types, and a function modifier that allows calls to and from Pascal programs and the Macintosh Interface Libraries. The language has built-in support for the Standard Apple Numeric Environment (SANE). It recognizes the SANE data types, uses SANE for all C floating-point operations and conversions, and correctly handles NaNs (Not-a-Number) and infinities in comparisons and in ASCII-binary conversions. The language together with the SANE library support comprise a scrupulously conforming implementation of extended-precision IEEE Standard 754 floating-point arithmetic. Furthermore, source programs written using only float and double types and standard C operations compile and run without modification. These and other extensions are discussed in detail below.

1.1 Data Types

The table below lists the arithmetic and pointer types available in Workshop C, and the number of bits allocated for variables of these types. Types int and longint, which are identical in this implementation, represent 32-bit integers. Pointers also require 32 bits. Enumeration types are allocated either 8, 16, or 32 bits, depending on the range of the enumeration literal values.

Data Type	Bits	Description
char	8	range –128 to 127
unsigned char	8	range 0 to 255
short	16	range –32,768 to 32,767
unsigned short	16	range 0 to 65,535
int	32	range –2,147,483,648 to 2,147,483,647

unsigned int	32	range 0 to 4,294,967,295
long	32	range –2,147,483,648 to 2,147,483,647
unsigned long	32	range 0 to 4,294,967,295
enum	8, 16 or 32	depends on the range of the enumeration literals
*	32	pointer types
float	32	IEEE single-precision floating point
double	64	IEEE double-precision floating point
comp	64	SANE signed integral values
extended	80	IEEE extended-precision floating point

1.2 Type Void

Type void has no values and no operators. Type void may be used as a type specifier in function declarations to indicate that the function has no meaningful return value. Specifying type void in Pascal-compatible function declarations reduces the number of instructions generated in calling the function. (See Section 1.10, Pascal-Compatible Functions.)

1.3 Type Enum

Type enum is a type analogous to the enumeration types of Pascal. Its syntax is similar to that of the struct and union declarations:

enum-specifier:
  enum { *enum-list* }
  enum *identifier* { *enum-list* }
  enum *identifier*

enum-list:
  enumeration-declaration
  enumeration-declaration , *enum-list*

enumeration-declaration:
  identifier
  identifier = *constant-expression*

The first identifier in **enum-specifier**, like the structure tag in a struct-specifier, names a particular enumeration. For example,

```
enum color {chartreuse, burgundy, claret, winedark};
. . .
enum color *cp, col;
```

This enumeration makes **color** the enumeration tag of a type describing various colors and then declares **cp** as a pointer to an object of that type and **col** as an object of that type.

The identifiers in **enum-list** are declared as constants and may appear wherever constants are required. If no enumerators with a **constant-expression** appear, the values of the constants begin at 0 and increase by 1 as the declaration is read from left to right. An enumerator with a **constant-expression** gives the associated identifier the value indicated; subsequent identifiers continue the progression by 1 from the assigned value.

Enumeration tags and constants must be unique. They are drawn from the set of ordinary identifiers, unlike structure tags and members. Objects of a given enumeration type have a type distinct from objects of all other types.

1.4 Register Variables

The compiler allocates automatic variables in registers whenever possible. Register variables will be assigned to registers before other automatic variables. Enumeration, character, integer, and pointer variables qualify for register allocation unless their address is taken with the & operator. Floating-point variables are not allocated to registers.

Several data and address registers are available for use as automatic variables. The exact number depends on the calling conventions being used. The number of variables allocated to registers may exceed the total number of registers. Several variables whose useful lifetimes do not overlap may be assigned to the same register. Often all of the eligible variables within a function will reside in registers, rather than on the stack.

1.5 Structures

Structures may be assigned, passed as parameters, and returned as function results. The left and right sides of a structure assignment must have the same type. Similiarly, actual and formal parameters must have identical types. Other plausible operators, such as equality comparison, have not been implemented.

Warning: In functions that return structures, if an interrupt occurs during the return sequence and the same function is called reentrantly during the interrupt, the value returned from the first call may be corrupted. The problem can occur only in the presence of interrupts. Recursive calls are quite safe.

1.6 Return, Newline, and Vertical Tab

The Return character, rather than newline, is represented by **\n** (a backslash character followed by a lowercase n). Return is the usual line termination character on the Macintosh. Return can also be represented by **\r**. Vertical tab is represented by **\v**.

1.7 __LINE__ and __FILE__

__LINE__ is a predefined preprocessor symbol whose value is the current line number within the current source file. __FILE__ is a similiar symbol whose value is a character string consisting of the current file name. Both symbols begin and end with two underscore characters.

1.8 Predefined Symbols

The symbols **MC68000, mc68000, m68k, ghs,** and **macintosh** are predefined. Each of the symbols has the value 1, as if a statement of this form had appeared at the beginning of the source code:

```
    #define MC68000 1
```

1.9 Standard Apple Numeric Environment Extensions

The Standard Apple Numeric Environment (SANE) is an extended-precision version of the IEEE Standard for Binary Floating-Point Arithmetic (754), together with an extra data type and basic functions for application development. Workshop C supports this environment. Much of SANE is provided through the run-time library sanelib and its include file sane.h. However, to use extended-precision arithmetic efficiently and effectively, and to handle IEEE NaNs (Not-a-Number) and infinities, some extensions to standard C are required.

A change from double to extended as the basic floating-point type is the most salient change to standard C. Since C was originally developed on the DEC PDP-11, the PDP-11 architecture is reflected in standard C in the use of float and double as floating-point types, with double as the basic type: floating-point expressions are evaluated to double, anonymous variables are double, and floating-point parameters and function results are passed as doubles. However, the low-level SANE arithmetic (as well as the floating-point chips Intel 8087, Motorola 68881, and Zilog Z8070) evaluates arithmetic operations to the range and precision of an 80-bit extended type. Thus, extended naturally replaces PDP-11 double as the basic arithmetic type for computing purposes. The types float (IEEE single), double, and comp serve as space-saving storage types, just as float does in conventional C.

The IEEE Standard specifies two special representations for its floating-point formats: NaNs (Not-a-Number) and infinities. Workshop C expands the syntax for I/O to accommodate NaNs and infinities, and includes the treatment of NaNs in relationals as required by the IEEE Standard.

The SANE extensions to standard C are backward compatible: programs written using only float and double floating-point types and standard C operations compile and run without modification. SANE does not affect integer arithmetic.

The *Apple Numerics Manual* contains detailed documentation of the Standard Apple Numeric Environment.

1.9.1 Constants

Numeric constants that include floating-point syntax—a point (.) or an exponent field—or that lie outside the range of longint are of type extended. Decimal-to-binary conversion for numeric constants is done at compile time (and hence is governed by the default numeric environment; compare Section 1.9.6).

1.9.2 Expressions

The SANE types—float, double, comp, and extended—can be mixed in expressions with each other and with integer types in the same manner that float and double can in standard C. An expression consisting solely of a SANE-type variable, constant, or function is of type extended. An expression formed by subexpressions and an arithmetic operation is of type extended if either of its subexpressions is. Extended-type expressions are evaluated using extended-precision SANE arithmetic, with conversions to type extended generated automatically as needed. Parentheses in extended-type expressions are honored. Initialization of external and static variables, which may include expression evaluation, is done at compile time; all other evaluation of extended-type expressions is done at run time.

1.9.3 Comparisons

The result of a comparison involving a NaN operand is unordered. The usual trichotomy of numbers is expanded to less (<), greater (>), equal (==), and unordered. For example, the negation of "a less than b" is not "a greater than or equal to b" but "(a greater than or equal to b) OR (a and b unordered)".

1.9.4 Functions

A numeric actual parameter passed by value is an expression and hence is of extended or integer type. All extended-type arguments are passed as extendeds. Similarly, all results of functions declared float, double, comp, or extended are returned as extendeds.

1.9.5 Input/Output

In addition to the usual syntax accepted for numeric input, the Standard C Library function *scanf* recognizes "INF" as infinity and "NAN" as a NaN. NAN may be followed by parentheses, which may contain an integer (a code indicating the NaN's origin). INF and NAN are optionally preceded by a sign and are case insensitive. *Scanf* specifiers for SANE types extend standard C as follows: conversion characters **f**, **e**, and **g** indicate type float; **lf**, **le**, and **lg** indicate type double; **mf**, **me**, and **mg** indicate type comp; and **ne**, **nf**, and **ng** indicate type extended.

The Standard C Library function *printf* writes infinities as [–]INF and NaNs as [–]NAN(ddd), where ddd is the NaN code.

1.9.6 Numeric Environment

The numeric environment refers to rounding direction, rounding precision, halt enables, and exception flags. IEEE Standard defaults—rounding to nearest, rounding to extended precision, and all halts disabled—are in effect for compile-time arithmetic (including decimal-to-binary conversion). Each program begins with these defaults and with all exception flags clear. Functions for managing the environment are included in the library sanelib. The compiler, in optimizing, will not change any part of the numeric environment, including the exception-flag setting which is a side effect of arithmetic operations.

1.9.7 SANE Library

The SANE library rounds out the IEEE Standard implementation and provides the basic tools for developing a wide range of applications. The SANE library includes the following:

logarithmic, exponential, and trigonometric functions
financial functions
random number generation
binary-decimal conversion
numeric scanning and formatting
environment control
other functions required or recommended by the IEEE Standard

Additional information can be found under the SANE entry in the Macintosh Interface Libraries section.

1.9.8 SANE Programming

Workshop C's automatic use of the extended type produces results that are generally better than those of other C systems. Extended precision yields more accuracy and extended range avoids unnecessary underflow and overflow of intermediate results. The programmer can further exploit the extended type by declaring all floating-point temporary variables to be type extended. This is both time- and space-efficient, since it reduces the number of automatic conversions among types. External data should be stored in one of the three smaller SANE types (float, double, or comp), not only for economy but also because the extended format may vary among SANE implementations. As a general rule, use float, double, or comp data as program input; extended arithmetic for computations; and float, double, or comp data as program output.

In many instances, IEEE arithmetic allows simpler algorithms than were possible without IEEE arithmetic. The handling of infinities enlarges the domain of some formulas. For example, $1+1/x^2$ computes correctly even if x^2 overflows. Running with halts disabled (the default), a program will never crash due to a floating-point exception. Hence by monitoring exception flags a program can test for exceptional cases after the fact. The alternative of screening out bad input is often infeasible, sometimes impossible.

1.10 Pascal-Compatible Functions

The function-calling conventions used by Workshop C and Pascal differ radically in the order of parameters on the stack, the type coercions applied to parameters, the location of the return result, and the number of scratch registers. C has been extended to allow function calls between these languages. The specifier pascal in a function declaration or definition indicates a Pascal-compatible function.

1.10.1 Pascal-Compatible Function Declarations

A function or procedure written in Pascal (or written in assembly language following Pascal calling conventions) can be called from C. A Pascal-compatible external function declaration begins with the **pascal** specifier, contains the usual type specifiers, function name, and parameter list, and must also contain declarations for the parameters, followed by the word **extern**. Parameters whose declarations are omitted are assumed to be type int. For example, the C function declaration

```
pascal void DrawText(textBuf,firstByte,byteCount)
   Ptr textBuf;
   short firstByte,byteCount;
   extern;
```

would allow a C program to call the procedure DrawText defined in Pascal as follows:

```
PROCEDURE DrawText(textBuf: Ptr; firstByte,byteCount: INTEGER);
```

Pascal-compatible function declarations are used in the Macintosh Interface Libraries to allow C programs to directly call Macintosh library routines that use Pascal calling conventions. The word **extern** may be followed by a constant, which is interpreted as a 16-bit 68000 instruction that replaces the usual subroutine call (JRS) instruction in the calling sequence. This allows direct traps to the Macintosh ROM. For example:

```
pascal void OpenPort(port)
   GrafPtr port;
   extern 0xA86F;
```

1.10.2 Pascal-Compatible Function Definitions

A function definition (the actual function), like a function declaration, can also be preceded by the **pascal** specifier. The function then adheres to Pascal-compatible calling conventions and can be called from Pascal. For example, the following C function can be called from Pascal:

```
pascal void MyText(byteCount,textAddr,numer,denom)
   short byteCount;
   Ptr textAddr;
   Point numer,demon;
{
   ...
}
```

The corresponding Pascal function declaration would be

```
PROCEDURE MyText(bytecount: INTEGER; textAddr: Ptr; numer,denom: Point);
```

For compatibility with Pascal and assembly language, the compiler converts the names of Pascal-compatible functions to uppercase before writing them to the object file. When they are called in C programs, these routines should be capitalized exactly as they were declared in C. Pascal-compatible functions whose names differ only in their capitalization will become duplicate declarations when their names are converted to uppercase by the compiler; therefore such names should be avoided.

1.10.3 Parameter and Result Types

C and Pascal support different data types. Therefore when writing a Pascal-compatible function declaration in C, a translation of the parameter types and function-result type (from Pascal to C) is required. Often this translation is trivial, but other cases are surprising.

The table below summarizes this translation. Find the Pascal parameter or result type in the first column. Use the equivalent C type found in the second column when declaring the function in C. Comments in the table point out unusual cases which may require special attention.

Pascal Parameter or Result Type	C Equivalent	Comments
boolean	Boolean	Boolean is defined in file types.h as **enum {false,true}**.
var boolean	Boolean *	In C, **false** is zero and **true** is often considered nonzero.
boolean result	Boolean	In Pascal, **false** is zero and **true** is one.
enumeration (<128 or >255 literals)	enum	Use identical ordering of the enumeration literals.
enumeration (128 to 255 literals)	short	Pascal passes enumerations with 128 or more literals as words.
var enumeration (<128 or >255 literals)	enum *	
var enumeration (128 to 255 literals)	short *	
enumeration result (<128 or >255 literals)	enum	
enumeration result (128 to 255 literals)	short	
char	short	Surprise! Pascal passes chars as 16-bit values.
var char	char *	
char result	short	
integer	short	16-bit signed values.
var integer	short *	
short result	short	
longint	int or long	32-bit signed values.
var longint	int * or long *	
longint result	int or long	
real	extended *	Pascal passes real parameters as extended by address.
var real	float *	
real result	float	Pascal returns real results by value.
double	extended *	Pascal passes double parameters as extended by address.
var double	double *	
double result	double	The caller supplies the address of the double result.
comp	extended *	Pascal passes comp parameters as extended by address.

var comp	comp *	
comp result	comp	The caller supplies the address of the comp result.
extended	extended *	Pascal passes extended parameters by address.
var extended	extended *	
extended result	extended	The caller supplies the address of the extended result.
pointer	pointer	32-bit addresses.
var pointer	pointer *	
pointer result	pointer	
array (1 or 2 bytes)	short	Pascal passes small arrays by value.
array (3 or 4 bytes)	int or long	
array (5 or more bytes)	array	Pascal passes larger arrays by address.
var array	array	
array result	---	C does not allow array results.
record (1 to 4 bytes)	struct	Pascal passes small records by value.
record (5 or more bytes)	struct *	Pascal passes larger records by address.
var record (any size)	struct *	
record result (1 or 2 bytes)	short	Pascal returns small records by value.
record result (3 or 4 bytes)	int or long	
record result (1 or 2 bytes)	struct	The caller supplies the address of the record result.
set (1 to 7 elements)	char	Pascal passes sets with 1 to 7 elements as bytes.
set (8 to 16 elements)	short	Pascal passes sets with 8 to 16 elements as words.
set (≥17 elements)	struct	Pascal also passes larger sets by value.
var set (1 to 7 elements)	char *	
var set (8 to 16 elements)	short *	
var set (≥17 elements)	struct *	
set result (1 to 7 elements)	char	Pascal returns small sets by value.
set result (8 to 16 elements)	short	
set result (≥17 elements)	struct	The caller supplies the address of the set result.

2. Implementation-Specific Language Details

A number of details in any language definition are left to the discretion of its individual implementations. Most programs do not rely on these details and therefore yield the same results on the various implementations. However, knowledge of the major differences between implementations can help avoid reliance on language semantics that vary from implementation to implementation. This section explains several areas of the language definition that are specific to Workshop C.

2.1 Byte Ordering

On the MC68000, the microprocessor used in the Macintosh, the least significant byte of a short or long integer has the highest memory address. This byte ordering is also used on IBM/370 and Z8000 processors. The PDP-11 family, VAX, 8086, and NS16000 use a different ordering. Programs that rely on the order of the bytes within words and longs will not work correctly on both classes of machines.

2.2 Memory-Allocation Characteristics

The Workshop C compiler optimizes memory allocation in various ways. Automatic variables (locals) are allocated in registers whenever possible. Static and global variables are not necessarily allocated in the order in which they are specified. (However, the order of fields within records is preserved.) Static variables may be allocated as if they were automatic if their values are always set before being referenced. Automatic and static variables that are never used may not be allocated at all. Programs should not rely on the compiler's allocation algorithms.

2.3 Unsigned Char and Unsigned Short

Types unsigned char and unsigned short are supported by the Workshop C compiler and by many implementations of PCC, although they are not required by the basic C language definition. The VAX implementation of PCC and the Workshop C compiler differ in the way they evaluate expressions involving these types. For example, the negation operator subtracts an unsigned short from 2^{16} under PCC (this seems like a bug), and from 2^{32} under Workshop C.

2.4 Bit Fields

Workshop C provides bit fields that are unsigned, as do all MC68000 versions of PCC of which we are aware. However, VAX implementations of C may support signed bit fields. In the following example, implementations using unsigned bit fields will set i to 3; implementations using signed bit fields will set i to –1.

```
struct {int  field:2;}  x;
x.field  =  3;
i  =  x.field;
```

2.5 Evaluation Order

Workshop C does not define the evaluation order of certain expressions. Expressions with side effects, such as function calls and the "++" and "– –" operators, may yield different results on different machines or with different compilers. Specifically, when a variable is modified as a side effect of an expression's evaluation and the variable is also used at another point in the same expression, the value used may be either the value before modification or the value after modification.

Programs that rely on the order of evaluation in these situations are in error. The function call **f(i,i++)** is an example of an expression whose value is undefined.

2.6 Case Statements

Some implementations of C, including PCC, allow cases of a switch statement to be nested within compound statements. Workshop C considers this an error. The following switch statement compiles using PCC but generates an error message using the Workshop C compiler. The error is that **case 2:** is within the **if** statement.

```
switch  (i)  {
case 1:
   if  (j)  {
      case  2:
         i  = 3;
   }
}
```

3. Implementation Notes

The definition of C specifies the meaning of C programs. Implementations vary considerably in how they choose to implement this definition. Certain implementation details are of interest to assembly language programmers who want to write functions that either call or are called from C. This section provides those details.

3.1 C Calling Conventions

Workshop C uses two different function-calling conventions. This section describes the usual C calling conventions. The Pascal-compatible calling conventions are described in section 3.2.

3.1.1 Parameters

Parameters to C functions are evaluated from right to left and are pushed onto the stack in the order they are evaluated. Characters, integers, and enumeration types are passed as sign-extended 32-bit values. Pointers and arrays are passed as 32-bit addresses. Types float, double, comp, and extended are passed as extended 80-bit values. Structures are also passed on the stack. Their size is rounded up to a multiple of 16 bits (2 bytes). If rounding occurs, the unused storage has the highest memory address. The caller removes the parameters from the stack.

3.1.2 Function Results

Characters, integers, enumeration types, and pointers are returned as sign-extended 32-bit values in register D0. Types float, double, comp, and extended are returned as extended values in registers D0, D1, and A0. The low-order 16 bits of D0 contain the sign and exponent bits; register D1 contains the high-order 32 bits of the significand; register A0 contains the low-order 32 bits of the significand. Structure values are returned as a 32-bit pointer in register D0. The pointer contains the address of a static variable into which the result is copied before returning. This implementation of structure function results is not reentrant.

3.1.3 Register Conventions

Registers D0, D1, A0, and A1 are scratch registers and are not preserved by C functions. All other registers are preserved. Register A5 is the global frame pointer, register A6 is the local frame pointer, and register A7 is the stack pointer. Local stack frames are not necessarily created for simple functions.

3.2 Pascal-Compatible Calling Conventions

This section describes the conventions used for calling Pascal functions from C and for functions written in C that use Pascal-compatible calling conventions. These conventions differ from the usual C calling conventions defined in section 3.1; they also differ from the calling conventions used by the Pascal compiler.

3.2.1 Parameters

Parameters to Pascal-compatible functions are evaluated left to right and are pushed onto the stack in the order they are evaluated. Characters and enumeration types whose literal values fall in the range of types char or unsigned char are pushed as bytes. (This requires a 16-bit word on the stack. The value is in the high-order 8 bits; the low-order 8 bits are unused.) Short ints and enumeration types whose literal values fall in the range of types short or unsigned short are passed as 16-bit values. Ints, long ints, and the remaining enumeration types are passed as 32-bit values. Pointers and arrays are passed as 32-bit addresses. SANE types float, double, comp, and extended are passed as extended 80-bit values; however this doesn't correspond to the Pascal compiler's calling conventions, so a compiler warning is given. The table in Section 1.10.3 shows the recommended way to pass SANE-type values to Pascal. Structures are also passed by value on the stack, and also yield a compiler warning. Their size is rounded up to a multiple of 16 bits (2 bytes). If rounding occurs, the unused storage has the highest memory address. The function being called removes the parameters from the stack.

3.2.2 Results

Function results are returned on the stack. Stack space for the function result is reserved by the caller prior to pushing any parameters. Characters and enumeration types whose literal values fall in the range of types char or unsigned char are returned as bytes. (This requires a 16-bit word on the stack. The value is in the high-order 8 bits; the low-order 8 bits are unused.) Short ints and enumeration types whose literal values fall in the range of types short or unsigned short are returned as 16-bit values. Ints, long ints, and the remaining enumeration types are returned as 32-bit values. Pointers are returned as 32-bit addresses. Arrays may not be returned as function results. Results of type float are returned as 32-bit values. For structures and types double, comp, and extended the caller pushes the address of a structure, double, comp, or extended (respectively) in the function-result location on the stack. The procedure being called stores the result at this address. The caller removes the function results from the stack.

3.2.3 Register Conventions

Registers D0, D1, D2, A0, and A1 are scratch registers. Scratch registers are not preserved by Pascal-compatible functions. All other registers are preserved. Register A5 is the global frame pointer, register A6 is the local frame pointer, and register A7 is the stack pointer.

3.3 Compiler Limitations

On the Macintosh, the total size of all declared global variables, static variables, and string constants cannot exceed 32K bytes. Allocate large global arrays on the heap in order to avoid exceeding this limit.

It is impossible to compile very large functions on the Lisa because the compiler's internal data structures cannot fit in memory. As functions approach this limit, compilation time increases noticeably. This problem can be alleviated by eliminating unnecessary include files, reducing the number of global declarations, compiling large functions separately, and rewriting large functions as two or more smaller functions. In addition, renaming Workshop files system.debug and system.debug2, then rebooting, will give the compiler an additional 80K bytes of memory.

Static functions are not supported in this implementation of C due to limitations in the Lisa Workshop object-file definition. Use of static functions may result in this linker message:

```
*** Error - Reference to unknown module type. ***
```

Static functions can be eliminated by including the define

```
#define static
```

at the beginning of each compilation or source file.

4. Running the Compiler

The C compiler is named c.obj. To run the compiler, use the Lisa Workshop Run command, as shown in the example below. Source file names must end with the suffix ".c.text". The ".text" suffix need not be specified in response to the input file prompt. The error-listing file is the console by default. The output file (object code file) name is by default the source file name, with the ".text" suffix replaced by ".obj".

Compiler options may be specified by responding to any of the file name prompts by typing a question mark (?) and then pressing Return. The compiler will present a partial list of the available options (a complete list appears below). Enter the options you want, including minus sign (–) prefixes, on a single line separated by spaces. The compiler will then repeat the file name prompt.

The example below compiles the program mac/c/sample/qdsample.c.text, writing the error listing to the console and creating the object file mac/c/sample/qdsample.obj. The **–M** option is used to direct the code to segment **sample**. Prompts supplied by the Workshop and the compiler are shown in plain text. Default file names appear within the prompts in brackets. What you type is printed in boldface. "<Return>" means press the Return key.

```
r
Run what program? c <Return>
C-68/Lisa 1.8.3 Copyright (c) 1984, 1985 Green Hills Software, Inc.
Name of input file [.TEXT] ? <Return>
Useful options include:
–Mname          Compile into segment 'name'.
–Dname          Define 'name' with value 1.
–Dname=string   Define 'name' with value 'string'.
–Uname          Undefine 'name'.
–Iprefix        Add 'prefix' to include-file names.
–E              Write preprocessor output to file 'preproc.text'.
–g              Generate debug symbols in code.
Enter options separated by blanks and followed by <Return>
Options> –Msample <Return>
Name of input file [.TEXT] mac/c/sample/qdsample.c <Return>
Name of error listing file [-console] / [.TEXT] <Return>
Name of binary file [ mac/c/sample/qdsample.c] [.OBJ] <Return>
```

4.1 Compiler Options

The Workshop C compiler recognizes the following options:

Option	Description
–Mname	Name the object-code segment. Since a segment may not exceed 32K bytes, large programs require multiple segments with different names. The default segment name is 8 blanks.
–Dname	Define **name** to the preprocessor with the value 1. This is equivalent to writing **#define name 1** at the beginning of the source file.
–Dname=string	Define **name** to the preprocessor with the value "string". This is equivalent to writing **#define name string** at the beginning of the source file.
–Uname	Undefine the predefined preprocessor symbol **name**. This is equivalent to writing **#undef name** at the beginning of the source file.

–Istring	Include file names that do not start with "–" are searched for with the file name prefix "string". Multiple –I options can be specified. They will be searched in the order encountered. The include-file search rules are explained in detail in section 4.2.
–w	Suppress compiler warning messages. By default, warnings are written to the error listing file.
–E	Do not compile the program. Instead, write the output of the preprocessor to file preproc.text. This option is useful for debugging preprocessor macros.
–C	Include comments with the preprocessor output. By default, comments are not written to the preprocessor output.
–g	Generate 8-byte function names embedded in the code for the debugger; generate frame pointers in A6 (**LINK A6,x ... UNLK A6**).
–ga	Generate stack frame pointers in A6 (**LINK A6,x ... UNLK A6**) for all functions.
–X6	Use **MOVE #0,x** instructions rather than **CLR x** instructions for nonstack addresses. This may be useful when writing device drivers.
–X9	Disable local (peephole) optimizations.
–X12	Generate 68010 code rather than 68000 code.
–Z6	Always allocate 32 bits for enumerated data types. The default is to allocate 8, 16, or 32 bits.
–X55	Make bit fields of type int, short, and char signed. The default is to make all fields unsigned.

4.2 Include-File Search Rules

The compiler automatically searches for include files in the manner described below. The first file successfully opened using these rules is included.

Include-File Name	Example	Search Rules

Begins with "–"	#include <-lower-consts.h>
	#include "-lower-consts.h"
		Use the specified name. None of the other search rules apply.

In angle brackets.	#include <ctype.h>	Begin the search using the prefix "mac/c/include/". Continue the search by prefixing the file name with strings supplied as –I options.
In double quotes.	#include "constants.h"	Begin the search using the prefix (up to the final "/") of the source file that contains the include. Continue by prefixing the file name with strings supplied as –I options. Finally, try the include-file name without any prefix.

5. Files to Link With

Workshop C programs should be linked using the linker, linker.obj. Programs to be executed on the Macintosh must be linked with one or more of the object files listed below. Programs must be linked with the object files that correspond to the libraries the program uses.

If no Standard C Library functions are called, link with:

mac/c/lib/runtime.obj	Execution starting point.

If Standard C Library functions are called, link with one of the following:

mac/c/lib/cruntime.obj	Doesn't prompt for *argv* and *argc* at the start of execution.
mac/c/lib/cruntimeargv.obj	Prompts for *argv* and *argc*.

If Standard C Library functions are called, also link with:

mac/c/lib/stdclib.obj	Standard C Library.
mac/c/lib/stdprintf.obj	Standard C Library *printf* functions.
mac/c/lib/math.obj	Standard C Library math functions.

If SANE library functions are called or floating-point arithmetic is used, also link with:

mac/c/lib/sanelib.obj	SANE numerics.

If Macintosh Interface Libraries functions are called, also link with:

mac/c/lib/interface.obj	Macintosh Interface Libraries.

If Macintosh Interface Libraries printing functions are called, also link with one of the following:

obj/prlink.obj	High-level printing interfaces.
obj/prscreen.obj	Low-level printing interfaces.

6. Standard C Library

This section describes the Standard C Library provided with Workshop C. After an introductory discussion of error-number conventions, the section is arranged alphabetically by function name or category. The Library Index is an index of the defines, types, enumeration literals, global variables, and functions defined here and in the Macintosh Interface Libraries.

Remember that identifiers in C are case sensitive and should be spelled *exactly as shown in the synopsis*. Following English usage, we've capitalized the first letter of a lowercase identifier when it appears at the beginning of a sentence.

NAME

Introduction to error numbers similar to those in UNIX operating systems.

SYNOPSIS

```
#include <errno.h>

extern int errno;
```

DESCRIPTION

Many of the Standard C Library functions have one or more error returns. An error condition is indicated by an otherwise impossible returned value. This is almost always –1; see descriptions of individual functions for details. An error number is also made available in the external variable *errno*. *Errno* is not cleared on successful calls, so it should be tested only after an error has been indicated.

The error name appears in brackets following the text in a library function description; for example,

> "The next attempt to write a nonzero number of bytes will signal an error. [ENOSPC]"

Not all possible error numbers are listed for each library function because many errors are possible for most of the calls. Some UNIX operating system error numbers do not apply to Macintosh and are not documented in this manual.

Here is a list of the error numbers and their names as defined in the <errno.h> file:

1 [EPERM] **Not owner**
Typically this error indicates an attempt to modify a file in a way that is not permitted.

2 [ENOENT] **No such file or directory**
This error occurs when a file whose filename is specified does not exist or when one of the directories in a pathname does not exist.

5 [EIO] **I/O error**
Some physical I/O error has occurred. This error may in some cases be signaled on a call following the one to which it actually applies.

6 [ENXIO] **No such device or address**
I/O on a special file refers to a subdevice that does not exist, or the I/O is beyond the limits of the device. This error may also occur when, for example, no disk is present in a drive.

9 [EBADF] **Bad file number**
Either a file descriptor does not refer to an open file, or a read (or write) request is made to a file that is open only for writing (or reading).

12 **`[ENOMEM]` Not enough space**
The system ran out of memory while the library call was executing.

13 **`[EACCES]` Permission denied**
An attempt was made to access a file in a way forbidden by the protection system.

17 **`[EEXIST]` File exists**
An existing file was mentioned in an inappropriate context—e.g., `open(file, O_CREAT+O_EXCL)`.

19 **`[ENODEV]` No such device**
An attempt was made to apply an inappropriate system call to a device—e.g., read a write-only device.

20 **`[ENOTDIR]` Not a directory**
An object that is not a directory was specified where a directory is required—e.g., in a path prefix.

21 **`[EISDIR]` Is a directory**
An attempt was made to write on a directory.

22 **`[EINVAL]]` Invalid argument**
Some invalid argument was provided to a library function.

23 **`[ENFILE]` File table overflow**
The system's table of open files is full, so temporarily a call to open() cannot be accepted.

24 **`[EMFILE]` Too many open files**
No program may have more than 20 file descriptors open at a time.

28 **`[ENOSPC]` No space left on device**
During a write() to an ordinary file, there is no free space left on the device.

29 **`[ESPIPE]` Illegal seek**
An lseek() was issued incorrectly.

30 **`[EROFS]` Read-only file system**
An attempt to modify a file or directory was made on a device mounted for read-only access.

NOTE

Calls that interface to the Macintosh I/O system—e.g., open(), close(), read(), write(), ioctl(), and others—set the external variable *MacOSErr* as well as *errno*. This manual documents only *errno* values. The equivalent Macintosh ROM error-return values set in *MacOSErr* are documented in *Inside Macintosh.* The appropriate include file for most values of *MacOSErr* is <files.h>.

NAME

abs—return integer absolute value

SYNOPSIS

```
int abs (i)
int i;
```

DESCRIPTION

Abs returns the absolute value of its integer operand.

NOTE

The absolute value of the negative integer with largest magnitude is undefined.

SEE ALSO

`floor().`

NAME

atof—convert ASCII string to floating-point number

SYNOPSIS

```
extended atof (nptr)
char *nptr;
```

DESCRIPTION

Atof converts a character string pointed to by *nptr* to an extended-precision floating-point number. The first unrecognized character ends the conversion. *Atof* recognizes an optional string of white-space characters (blanks or tabs), then an optional sign, then a string of digits optionally containing a decimal point, then an optional "e" or "E" followed by an optionally signed integer. If the string begins with an unrecognized character, *atof* returns a NaN.

DIAGNOSTICS

Atof honors the floating-point exception flags—invalid operation, underflow, overflow, divide by zero, and inexact—as prescribed by the Standard Apple Numeric Environment (SANE).

SEE ALSO

`scanf(), str2dec(), dec2num().`
Apple Numerics Manual.

NAME

atoi—convert string to integer

SYNOPSIS

```
int atoi (str)
char *str;
```

DESCRIPTION

Atoi returns as an integer the decimal value represented by the character string *str*. The string is scanned up to the first nondigit character other than an optional leading minus sign (–). Leading white-space characters (blanks and tabs) are ignored.

NOTE

Overflow conditions are ignored.

SEE ALSO

`atof(), scanf().`

NAME

close—close a file descriptor

SYNOPSIS

```
int close (fildes)
int fildes;
```

DESCRIPTION

Fildes is a file descriptor obtained from a creat() or open() call. *Close* closes the file descriptor indicated by *fildes*.

Close fails if *fildes* is not a valid open file descriptor. [EBADF]

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.

SEE ALSO

`creat(), open().`

NAME

toupper, tolower, _toupper, _tolower, toascii
—translate characters

SYNOPSIS

```
#include <ctype.h>

int toupper (c)
int c;

int tolower (c)
int c;

int _toupper (c)
int c;

int _tolower (c)
int c;

int toascii (c)
int c;
```

DESCRIPTION

Toupper and *tolower* have as domain the range of getc(): the integers from −1 through 255. If the argument of *toupper* represents a lowercase letter, the result is the corresponding uppercase letter. If the argument of *tolower* represents an uppercase letter, the result is the corresponding lowercase letter. All other arguments in the domain are returned unchanged.

_Toupper and *_tolower* are macros that accomplish the same thing as *toupper* and *tolower* but have restricted domains and are faster. *_Toupper* requires a lowercase letter as its argument; its result is the corresponding uppercase letter. *_Tolower* requires an uppercase letter as its argument; its result is the corresponding lowercase letter. Arguments outside the domain cause undefined results.

Toascii yields its argument with all bits turned off that are not part of a standard ASCII character; it is intended for compatibility with other systems.

SEE ALSO

`ctype, getc().`

NAME

creat—create a new file or rewrite an existing file

SYNOPSIS

```
int creat (path)
char *path;
```

DESCRIPTION

Creat creates a new file or prepares to rewrite an existing file named by the pathname pointed to by *path*. If the file exists, the length of its data fork is truncated to 0.

Creat is equivalent to `open(path,O_WRONLY|O_TRUNC)`.

Upon successful completion, a nonnegative integer (the file descriptor) is returned and the file is open for writing. The file pointer is set to the beginning of the file. A maximum of 20 files may be open at a given time.

RETURN VALUE

Upon successful completion, a nonnegative integer (the file descriptor) is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

`close(), lseek(), open(), read(), write().`

NAME

isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, isgraph, iscntrl, isascii
—classify characters

SYNOPSIS

```
#include <ctype.h>

int isalpha (c)
int c;

. . .
```

DESCRIPTION

These macros classify character-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. *Isascii* is defined on all integer values; the rest are defined only where *isascii* is true and on the single non-ASCII value EOF (–1).

isalpha	*c* is a letter.
isupper	*c* is an uppercase letter.
islower	*c* is a lowercase letter.
isdigit	*c* is a digit [0-9].
isxdigit	*c* is a hexadecimal digit [0-9], [A-F], or [a-f].
isalnum	*c* is alphanumeric (letter or digit).
isspace	*c* is a space, tab, return, new line, vertical tab, or form feed.
ispunct	*c* is a punctuation character (neither control nor alphanumeric).
isprint	*c* is a printing character, code 040 (space) through 0176 (tilde).
isgraph	*c* is a printing character, similar to *isprint* except false for space.
iscntrl	*c* is a delete character (0177) or an ordinary control character (less than 040).
isascii	*c* is an ASCII character, code less than 0200.

DIAGNOSTICS

If the argument to any of these macros is not in the domain of the function, the result is undefined.

NAME

ecvt, fcvt—convert floating-point number to string

SYNOPSIS

```
char *ecvt(value, ndigit, decpt, sign)
extended value;
int ndigit, *decpt, *sign;

char *fcvt(value, ndigit, decpt, sign)
extended value;
int ndigit, *decpt, *sign;
```

DESCRIPTION

Ecvt converts *value* to a null-terminated string of *ndigit* digits and returns a pointer to this string as the function result. The low-order digit is rounded.

The decimal point is not included in the returned string. The position of the decimal point is indicated by *decpt*, which indirectly stores the position of the decimal point relative to the returned string. If the int pointed to by *decpt* is negative, the decimal point lies to the left of the returned string. For example, if the string is "12345" and *decpt* points to an int of 3, the value of the string is 123.45; if *decpt* points to –3, the value of the string is .00012345 .

If the sign of the converted value is negative, the word pointed to by *sign* is nonzero; otherwise it is zero.

Fcvt and *ecvt* provide fixed-point output in the style of Fortran F-format output, with the following difference in the interpretation of *ndigit*:

> In *fcvt*, *ndigit* specifies the number of digits to the right of the decimal point.
> In *ecvt*, *ndigit* specifies the number of digits in the string.

NOTE

The string pointed to by the function result is static data whose content is overwritten by each call.

SEE ALSO

`printf(), num2dec(), dec2str.`
Apple Numerics Manual.

NAME

exit, _exit—terminate the current application

SYNOPSIS

```
void exit(status)
int status;
void _exit(status)
int status;
```

DESCRIPTION

Exit terminates the current application, closing all of the open file descriptors. It also causes stdio() cleanup actions before the application terminates.

The function _*exit* circumvents all cleanup.

NAME

exp, log, log10, pow, sqrt
—exponential, logarithm, power, square-root functions

SYNOPSIS

```
#include <math.h>

extended exp(x)
extended x;

extended log(x)
extended x;

extended log10(x)
extended x;

extended pow(x, y)
extended x, y;

extended sqrt(x)
extended x;
```

DESCRIPTION

Exp returns e^x, where e is the natural logarithm base.

Log returns the natural logarithm (base e) of x.

Log10 returns the logarithm base ten of x.

Pow returns x^y.

Sqrt returns the square root of x.

For special cases, these functions return a NaN or signed infinity as appropriate.

DIAGNOSTICS

These functions honor the floating-point exception flags—invalid operation, underflow, overflow, divide by zero, and inexact—as prescribed by the Standard Apple Numeric Environment (SANE).

SEE ALSO

`hypot(), sinh().`
Apple Numerics Manual.

NAME

fclose, fflush—close or flush a stream

SYNOPSIS

```
#include <stdio.h>

int fclose (stream)
FILE *stream;

int fflush (stream)
FILE *stream;
```

DESCRIPTION

Fclose causes any buffered data for *stream* to be written out; *stream* is then closed.

Fclose is performed automatically for all open files upon calling exit().

Fflush causes any buffered data for *stream* to be written out; *stream* remains open.

DIAGNOSTICS

These functions return 0 for success or EOF if an error was detected (such as trying to write to a file that has not been opened for writing).

SEE ALSO

`close(), exit(), fopen(), setbuf().`

NAME

ferror, feof, clearerr, fileno—stream status inquiries

SYNOPSIS

```
#include <stdio.h>

int feof (stream)
FILE *stream;

int ferror (stream)
FILE *stream;

void clearerr (stream)
FILE *stream;

int fileno (stream)
FILE *stream;
```

DESCRIPTION

Feof returns nonzero when end-of-file has previously been detected reading the named input stream; otherwise, it returns zero.

Ferror returns nonzero when an I/O error has previously occurred reading from or writing to the named stream; otherwise, it returns zero.

Clearerr resets the error indicator and end-of-file indicator to zero on the named stream.

Fileno returns the integer file descriptor associated with the named stream; see open().

NOTE

All these functions are implemented as macros; they cannot be declared or redeclared.

SEE ALSO

`open(), fopen().`

NAME

floor, ceil, fmod, fabs—floor, ceiling, mod, absolute value functions

SYNOPSIS

```
#include <math.h>

extended floor(x)
extended x;

extended ceil(x)
extended x;

extended fmod(x, y)
extended x, y;

extended fabs(x)
extended x;
```

DESCRIPTION

Floor returns the largest integer (as an extended-precision number) not greater than x.

Ceil returns the smallest integer not less than x.

Whenever possible, *fmod* returns the number f with the same sign as x, such that $x = iy + f$ for some integer i, and $|f| < |y|$. If y is zero, *fmod* returns a NaN.

Fabs returns $|x|$.

SEE ALSO

`abs(), rint(), setround().`
Apple Numerics Manual.

NAME

fopen, freopen, fdopen—open a stream

SYNOPSIS

```
#include <stdio.h>

FILE *fopen (filename, type)
char *filename, *type;

FILE *freopen (filename, type, stream)
char *filename, *type;
FILE *stream;

FILE *fdopen (fildes, type)
int fildes;
char *type;
```

DESCRIPTION

Fopen opens the file named by *filename* and associates a stream with it. *Fopen* returns a pointer to the FILE structure associated with the stream.

Filename points to a character string that contains the name of the file to be opened.

Type is a character string having one of the following values:

r	open for reading.
w	truncate or create for writing.
a	append: open for writing at end-of-file, or create for writing.
r+	open for update (reading and writing).
w+	truncate or create for update.
a+	append: open or create for update at end-of-file.

Freopen substitutes the named file for the open stream. The original stream is closed, regardless of whether the open ultimately succeeds. *Freopen* returns a pointer to the FILE structure associated with *stream.*

Freopen is typically used to attach the previously opened streams associated with *stdin*, *stdout*, and *stderr* to other files.

Fdopen associates a stream with a file descriptor by formatting a file structure from the file descriptor. Thus, *fdopen* can be used to access the file descriptors returned by open() or creat(). (These calls open files but do not return pointers to a FILE structure.) The type of stream must agree with the mode of the open file.

When a file is opened for update, both input and output may be done on the resulting stream. However, output may not be directly followed by input without an intervening fseek() or rewind(), and input may not be directly followed by output without an intervening fseek(), rewind(), or an input operation that encounters end-of-file.

When a file is opened for append (i.e., when *type* is "a" or "a+"), it is impossible to overwrite information already in the file. Fseek() may be used to reposition the file pointer to any position in the file, but when output is written to the file the current file pointer is disregarded. All output is written at the end of the file and causes the file pointer to be repositioned at the end of the output.

DIAGNOSTICS

Fopen and *freopen* return a null pointer on failure.

SEE ALSO

`open(), fclose().`

NAME

fread, fwrite—binary input/output

SYNOPSIS

```
#include <stdio.h>

int fread(ptr, size, nitems, stream)
char *ptr;
int size, nitems;
FILE *stream;

int fwrite(ptr, size, nitems, stream)
char *ptr;
int size, nitems;
FILE *stream;
```

DESCRIPTION

Fread copies *nitems* items of data from the named input stream into an array beginning at *ptr*. An item of data is a sequence of bytes (not necessarily terminated by a null byte) of length *size*. *Fread* stops appending bytes if an end-of-file or error condition is encountered while reading *stream* or if *nitems* items have been read. *Fread* leaves the file pointer in *stream*, if defined, pointing to the byte following the last byte read if there is one. *Fread* does not change the contents of *stream*.

Fwrite appends at most *nitems* items of data to the named output stream from the array pointed to by *ptr*. *Fwrite* stops appending when it has appended *nitems* items of data or if an error condition is encountered on *stream*. *Fwrite* does not change the contents of the array pointed to by *ptr*.

The variable *size* is typically `sizeof(*ptr)` where the pseudo-function *sizeof* specifies the length of an item pointed to by *ptr*. If *ptr* points to a data type other than char it should be cast into a pointer to char.

DIAGNOSTICS

Fread and *fwrite* return the number of items read or written. If *nitems* is zero or negative, no characters are read or written and zero is returned by both *fread* and *fwrite*.

SEE ALSO

`fopen(), getc(), gets(), printf(), putc(), puts(), read(), scanf(), stdio(), write().`

NAME

frexp, ldexp, modf—manipulate parts of floating-point numbers

SYNOPSIS

```
extended frexp(value, eptr)
extended value;
int *eptr;

extended ldexp(value, exp)
extended value;
int exp;

extended modf(value, iptr)
extended value, *iptr;
```

DESCRIPTION

Every nonzero number can be written uniquely as $x * 2^n$, where the mantissa (fraction) x is in the range $0.5 \leq |x| < 1.0$, and the exponent n is an integer. *Frexp* returns the mantissa of an extended value and stores the exponent indirectly in the location pointed to by *eptr*. Note that the mantissa here differs from the significand described in the *Apple Numerics Manual*, whose normal values are in the range $1.0 \leq |x| < 2.0$.

Ldexp returns the quantity $value * 2^{exp}$.

Modf returns the signed fractional part of *value* and stores the integral part indirectly in the location pointed to by *iptr*.

DIAGNOSTICS

Ldexp honors the floating-point exception flags—invalid operation, underflow, overflow, divide by zero, and inexact—as prescribed by the Standard Apple Numeric Environment (SANE).

SEE ALSO

`logb(), scalb().`
Apple Numerics Manual.

NAME

fseek, rewind, ftell—reposition a file pointer in a stream

SYNOPSIS

```
#include <stdio.h>

int fseek (stream, offset, ptrname)
FILE *stream;
long offset;
int ptrname;

void rewind (stream)
FILE *stream;

long ftell (stream)
FILE *stream;
```

DESCRIPTION

Fseek sets the position of the next input or output operation on the stream. The new position is at the signed distance *offset* bytes from the beginning, the current position, or the end of the file, when the value of *ptrname* is 0, 1, or 2, respectively.

`Rewind(stream)` is equivalent to `fseek(stream, 0L, 0)` except that no value is returned.

Fseek and *rewind* undo any effects of ungetc().

After *fseek* or *rewind*, the next operation on a file opened for update may be either input or output.

Ftell returns the offset of the current byte relative to the beginning of the file associated with the named stream.

DIAGNOSTICS

Fseek returns nonzero for improper seeks; otherwise it returns 0. An improper seek can be, for example, an *fseek* done on a file that has not been opened via fopen().

SEE ALSO

`lseek(), fopen().`

NAME

getc, getchar, fgetc, getw—get character or word from stream

SYNOPSIS

```
#include <stdio.h>

int getc(stream)
FILE *stream;

int getchar()

int fgetc(stream)
FILE *stream;

int getw(stream)
FILE *stream;
```

DESCRIPTION

Getc returns the next character (i.e., byte) from the named input stream. It also moves the file pointer, if defined, ahead one character in *stream*. *Getc* is a macro and therefore cannot be used if a function is necessary; for example, you cannot have a function pointer point to it.

Getchar returns the next character from the standard input stream, *stdin*. Like *getc*, *getchar* is a macro.

Fgetc behaves like*getc* but is a function. *Fgetc* runs more slowly than *getc* but takes less space per invocation.

Getw returns the next "word" (i.e., four bytes) from the named input stream so that the order of bytes in the stream corresponds to the order of byes in memory. *Getw* returns the constant EOF upon end-of-file or error. Since EOF is a valid integer value, feof() and ferror() should be used to check the success of *getw*. *Getw* increments the associated file pointer, if defined, to point to the next word. *Getw* assumes no special alignment in the file.

DIAGNOSTICS

These functions return the integer constant EOF at end-of-file or upon an error.

NOTE

Because it is implemented as a macro, *getc* treats incorrectly a stream argument with side effects. In particular, `getc(*f++)` doesn't work as you would expect. Use `fgetc(*f++)` instead.

SEE ALSO

`fclose(), ferror(), fopen(), fread(), gets(), putc(), scanf(), stdio().`

NAME

gets, fgets—get a string from a stream

SYNOPSIS

```
#include <stdio.h>

char *gets(s)
char *s;

char *fgets(s, n, stream)
char *s;
int n;
FILE *stream;
```

DESCRIPTION

Gets reads characters from the standard input stream, *stdin*, into the array pointed to by *s*, until a newline character is read or an end-of-file condition is encountered. The newline character is discarded and the string is terminated with a null character.

Fgets reads characters from the stream into the array pointed to by *s* until n–1 characters are read, or a newline character is read and transferred to *s*, or an end-of-file condition is encountered. The string is then terminated with a null character.

DIAGNOSTICS

If end-of-file is encountered and no characters have been read, no characters are transferred to *s* and a null pointer is returned. If a read error occurs, a null pointer is returned. Otherwise *s* is returned. (A read error will occur, for example, if you attempt to use these functions on a file that has not been opened for reading.)

SEE ALSO

`ferror(), fopen(), fread(), getc(), scanf(), stdio().`

NAME

hypot—Euclidean distance function

SYNOPSIS

```
#include <math.h>

extended hypot (x, y)
extended x, y;
```

DESCRIPTION

Hypot returns

sqrt (x * x + y * y)

taking precautions against unwarranted overflows.

DIAGNOSTICS

Hypot honors the floating-point exception flags—invalid operation, underflow, overflow, divide by zero, and inexact—as prescribed by the Standard Apple Numeric Environment (SANE).

SEE ALSO

`sqrt()`.
Apple Numerics Manual.

NAME

lseek—move read/write file pointer

SYNOPSIS

```
long lseek (fildes, offset, whence)
int fildes;
long offset;
int whence;
```

DESCRIPTION

Fildes is a file descriptor returned from a creat() or open() call. *Lseek* sets the file pointer associated with *fildes* as follows:

> If *whence* is 0, the pointer is set to *offset* bytes.
>
> If *whence* is 1, the pointer is set to its current location plus *offset.*
>
> If *whence* is 2, the pointer is set to the size of the file plus *offset.*

Upon successful completion, the resulting pointer location as measured in bytes from the beginning of the file is returned.

Lseek fails and the file pointer remains unchanged if one or more of the following are true:

> *Fildes* is not an open file descriptor. **[EBADF]**
>
> *Whence* is not 0, 1, or 2. **[EINVAL]**
>
> The resulting file pointer would be negative. **[EINVAL]**

Some devices are incapable of seeking. The value of the file pointer associated with such a device is undefined.

RETURN VALUE

Upon successful completion, a nonnegative integer indicating the file pointer value is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

NOTE

In previous versions of the Standard C Library, `tell(filedes)` was a function that returned the current file position. It is equivalent to the call `lseek(fildes,0L,1)`.

SEE ALSO

`creat(), open().`

NAME

malloc, free, realloc, calloc—main memory allocator

SYNOPSIS

```
char *malloc(size)
unsigned size;

void free(ptr)
char *ptr;

char *realloc(ptr, size)
char *ptr;
unsigned size;

char *calloc(nelem, elsize)
unsigned nelem, elsize;
```

DESCRIPTION

Malloc and *free* provide a simple general-purpose memory allocation package. The memory that is allocated for a program's use is known as the "storage arena." The storage arena expands as *malloc* is called.

Malloc returns a pointer to a block of at least *size* bytes suitably aligned for any use. The argument to *free* is a pointer to a block previously allocated by *malloc*; after *free* is performed this space is made available for further allocation.

Undefined results occur if the space assigned by *malloc* is overrun or if some random value is handed to *free.*

Malloc allocates the first sufficiently large contiguous reach of free space it finds. It calls NewPointer() to get more memory from the system when there is no suitable space already free.

Realloc changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block. The contents are unchanged up to the lesser of the new and old sizes. If no free block of *size* bytes is available in the storage arena, *realloc* asks *malloc* to enlarge the storage arena by *size* bytes and then moves the data to the new space.

Calloc allocates space for an array of *nelem* elements of size *elsize.* The space is initialized to zeros.

DIAGNOSTICS

Malloc, realloc, and *calloc* return a null pointer if there is no available memory or if the storage arena has been detectably corrupted by storing outside the bounds of a block. When this happens the block pointed to by *ptr* may have been destroyed.

NAME

memccpy, memchr, memcmp, memcpy, memset—memory operations

SYNOPSIS

```
char *memccpy(s1, s2, c, n)
char *s1, *s2;
int c, n;

char *memchr(s, c, n)
char *s;
int c, n;

int memcmp(s1, s2, n)
char *s1, *s2;
int n;

char *memcpy(s1, s2, n)
char *s1, *s2;
int n;

char *memset(s, c, n)
char *s;
int c, n;
```

DESCRIPTION

These functions operate efficiently on memory areas (arrays of characters bounded by a count, not terminated by a null character). They do not check for the overflow of any receiving memory area.

Memccpy copies characters from memory area *s2* into *s1*, stopping after the first occurrence of character *c* has been copied or after *n* characters have been copied, whichever comes first. It returns either a pointer to the character after the copy of *c* in *s1* or a null pointer if *c* was not found in the first *n* characters of *s2*.

Memchr returns either a pointer to the first occurrence of character *c* in the first *n* characters of memory area *s* or a null pointer if *c* does not occur.

Memcmp compares its arguments, looking at the first *n* characters only. It returns an integer less than, equal to, or greater than 0, depending on whether *s1* is less than, equal to, or greater than *s2* in the ASCII collating sequence.

Memcpy copies *n* characters from memory area *s2* to *s1*. It returns *s1*.

Memset sets the first *n* characters in memory area *s* to the value of character *c*. It returns *s*.

NOTE

Overlapping moves may yield unexpected results.

NAME

open—open for reading or writing

SYNOPSIS

```
#include <fcntl.h>

int open(path, oflag)
char *path;
int oflag;
```

DESCRIPTION

Path points to a pathname naming a file. *Open* opens a file descriptor for the named file and sets the file status flags according to the value of *oflag*. *Oflag* values are constructed by or-ing flags from the following list (only one of the first three flags below may be used):

O_RDONLY	Open for reading only.
O_WRONLY	Open for writing only.
O_RDWR	Open for reading and writing.
O_APPEND	If set, the file pointer is set to the end of the file prior to each write.
O_CREAT	If the file does not exist, it is created.
O_TRUNC	If the file exists, its length is truncated to 0; the mode and owner are unchanged.
O_EXCL	If O_EXCL and O_CREAT are set, *open* fails if the file exists.

Upon successful completion a nonnegative integer, the file descriptor, is returned. No application may have more than 20 file descriptors open simultaneously. The file pointer used to mark the current position within the file is set to the beginning of the file.

The named file is opened unless one or more of the following are true:

O_CREAT is not set and the named file does not exist. **[ENOENT]**

20 file descriptors are currently open. **[EMFILE]**

O_CREAT and O_EXCL are set and the named file exists. **[EEXIST]**

RETURN VALUE

Upon successful completion, a nonnegative integer (the file descriptor) is returned; otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

`close(), creat(), lseek(), read(), write().`

NAME

printf, fprintf, sprintf—print formatted output

SYNOPSIS

```
#include <stdio.h>

int printf(format [ , arg ] ... )
char *format;

int fprintf(stream, format [ , arg ] ... )
FILE *stream;
char *format;

int sprintf(str, format [ , arg ] ... )
char *str, format;
```

DESCRIPTION

Printf places formatted output on the standard output stream *stdout. Fprintf* places formatted output on the named output stream. *Sprintf* places formatted output, followed by the null character (\0), into the character array pointed to by *str*; it's your responsibility to ensure that enough storage is available. Each function returns the number of characters transmitted (not including the \0 in the case of *sprintf*), or a negative value if an output error was encountered.

Each of these functions converts, formats, and prints its args under control of the format. The format is a character string that contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching zero or more args. The results are undefined if there are insufficient args for the format. If the format is exhausted while args remain, the extra args are ignored.

Each conversion specification is introduced by the character %. After %, the following appear in sequence:

- Zero or more flag characters, which modify the meaning of the conversion specification.

- An optional decimal digit string specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded to the field width on the left (default) or right (if the left-adjustment flag has been given); see below for flag specification.

- A precision that gives the minimum number of digits to appear for the **d**, **o**, **u**, **x**, or **X** conversions, the number of digits to appear after the decimal point for the **e**, **E**, and **f** conversions, the maximum number of significant digits for the **g** and **G** conversions, or the maximum number of characters to be printed from a string in **s** conversion. The format of the precision is a period (.) followed by a decimal digit string; a null digit string is treated as zero.

- An optional **l** specifying that a following **d**, **o**, **u**, **x**, or **X** conversion character applies to a long-integer arg. The **l** option is ignored in this implementation since integers and long integers both require 32 bits.

- A character that indicates the type of conversion to be applied.

A field width or precision may be indicated by an asterisk (*) instead of a digit string. In this case, an integer arg supplies the field width or precision. The arg that is actually converted is not fetched until the conversion letter is seen; therefore, the args specifying field width or precision must appear immediately before the arg (if any) to be converted.

The flag characters and their meanings are:

–	The result of the conversion will be left-justified within the field.
+	The result of a signed conversion always begins with a sign (+ or –).
blank	If the first character of a signed conversion is not a sign, a blank will be prefixed to the result. This implies that if the **blank** and + flags both appear, the **blank** flag will be ignored.
#	This flag specifies that the value is to be converted to an "alternate form." For **c**, **d**, **s**, and **u** conversions, the flag has no effect. For **o** conversion, it increases the precision to force the first digit of the result to be a zero. For **x** (**X**) conversion, a nonzero result will have "0x" ("0X") prefixed to it. For **e**, **E**, **f**, **g**, and **G** conversions, the result will always contain a decimal point, even if no digits follow the point. (Normally, a decimal point appears in the result of these conversions only if a digit follows it.) For **g** and **G** conversions, trailing zeros in the fractional part will not be removed from the result (as they normally are).

The conversion characters and their meanings are:

d,o,u,x,X	The integer arg is converted to signed decimal (**d**), unsigned octal (**o**), decimal (**u**), or hexadecimal notation (**x** and **X**), respectively; the letters "abcdef" are used for **x** conversion and the letters "ABCDEF" for **X** conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is a null string.
f	The float, double, comp, or extended arg is converted to decimal notation in the style "[–]ddd.ddd", where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, it is assumed to be 6; if the precision is explicitly 0, no decimal point appears. Infinities are printed as "[–]INF" and NaNs are printed as "[–]NAN(ddd)" where ddd is a code indicating why the result is not a number.
e,E	The float, double, comp, or extended arg is converted in the style "[–]d.ddde±dd", where there is one digit before the decimal point

and the number of digits after it is equal to the precision; when the precision is missing, it is assumed to be 6; if the precision is zero, no decimal point appears. The **E** format code produces a number with "E" instead of "e" introducing the exponent. The exponent always contains at least two digits. Infinities are printed as "INF" and NaNs are printed as "[-]NAN(ddd)", where ddd is a code indicating why the result is not a number.

g,G The float, double, comp, or extended arg is printed in style **f** or **e** (or in style **E** in the case of a **G** format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style **e** is used only if the exponent resulting from the conversion is less than –4 or greater than the precision. Trailing zeros are removed from the result. A decimal point appears only if it is followed by a digit.

c The character arg is printed.

s The arg is taken to be a string (character pointer) and characters from the string are printed until a NULL character (\0) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed. If the string pointer arg has the value zero, the result is undefined. A null arg yields undefined results.

% Print a %; no argument is converted. In no case does a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by *printf* and *fprintf* are printed as if putc() had been called.

EXAMPLES

To print a date and time in the form "Sunday, July 3, 10:02", where weekday and month are pointers to null-terminated strings:

```
printf("%s, %s %d, %.2d:%.2d", weekday, month, day, hour, min);
```

To print pi to 5 decimal places:

```
printf("pi = %.5f", pi());
```

SEE ALSO

`dec2str, ecvt(), num2dec(), putc(), scanf(), stdio().`

NAME

putc, putchar, fputc, putw—put character or word on a stream

SYNOPSIS

```
#include <stdio.h>

int putc(c, stream)
char c;
FILE *stream;

int putchar(c)
char c;

int fputc(c, stream)
char c;
FILE *stream;

int putw(w, stream)
int w;
FILE *stream;
```

DESCRIPTION

Putc writes the character *c* to the output stream at the position pointed to by the file pointer, if one is defined. `Putchar(c)` is equivalent to `putc(c, stdout)`. *Putc* and *putchar* are macros.

Fputc behaves like *putc* but is a function rather than a macro. *Fputc* runs more slowly than *putc* but takes less space per invocation.

Putw writes the "word" *w* (i.e., four bytes) to the output stream at the position pointed to by the file pointer, if one is defined. *Putw* neither assumes nor causes special alignment in the file.

Output streams, with the exception of the standard error stream *stderr*, are by default buffered if the output refers to a file and line-buffered if the output refers to a window. *Stderr* is by default unbuffered, but use of freopen() causes it to become buffered or line-buffered. When an output stream is unbuffered information, it is queued for writing on the destination file or window as soon as written; when it is buffered, many characters are saved up and written as a block; when it is line-buffered, each line of output is queued for writing on the destination window as soon as the line is completed (i.e., as soon as a newline character is written or terminal input is requested). Setbuf() may be used to change the stream's buffering strategy.

DIAGNOSTICS

On success, these functions each return the value they have written. On failure, they return the constant EOF. This occurs if the file stream is not open for writing or if the output file cannot be grown. Because EOF is a valid integer, ferror() should be used to detect *putw* errors.

NOTE

Because it is implemented as a macro, *putc* treats incorrectly a stream argument with side effects. In particular, `putc(c, *f++);` doesn't work as you would expect. *Fputc* should be used instead.

SEE ALSO

fclose(), ferror(), fopen(), fread(), getc(), printf(), puts(), setbuf(), stdio().

NAME

puts, fputs—write a string to a stream

SYNOPSIS

```
#include <stdio.h>

int puts(s)
char *s;

int fputs(s, stream)
char *s;
FILE *stream;
```

DESCRIPTION

Puts writes the null-terminated string pointed to by *s*, followed by a newline character, to the standard output stream stdout.

Fputs writes the null-terminated string pointed to by *s* to the named output stream.

Neither function writes the terminating null character.

DIAGNOSTICS

Both routines return EOF on error. This occurs if the routines try to write on a file that has not been opened for writing.

NOTE

Puts appends a newline character while *fputs* does not.

SEE ALSO

`ferror(), fopen(), fread(), printf(), putc(), stdio().`

NAME

rand, srand—simple random-number generator

SYNOPSIS

```
int rand( )

void srand(seed)
unsigned seed;
```

DESCRIPTION

Rand uses a multiplicative congruential random-number generator with period 2^{32} that returns successive pseudorandom numbers in the range from 0 to $2^{15}–1$.

Srand can be called at any time to reset the random-number generator to a specific seed. The generator is initially seeded with a value of 1.

SEE ALSO

`randomx().`

NAME
read—read from file

SYNOPSIS
```
int read(fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;
```

DESCRIPTION
Fildes is a file descriptor obtained from a creat() or open() call.

Read attempts to read *nbyte* bytes from the file associated with *fildes* into the buffer pointed to by *buf.*

On devices capable of seeking, the read starts at a position in the file given by the file pointer associated with *fildes*. Upon return from *read,* the file pointer is incremented by the number of bytes actually read.

Nonseeking devices always read from the current position. The value of a file pointer associated with such a file is undefined.

Upon successful completion, *read* returns the number of bytes actually read and placed in the buffer; this number may be less than *nbyte* if the file is associated with a window or if the number of bytes left in the file is less than *nbyte* bytes. A value of 0 is returned when an end-of-file has been reached.

Read fails if *fildes* is not a valid file descriptor open for reading. **[EBADF]**

RETURN VALUE
Upon successful completion a nonnegative integer is returned indicating the number of bytes actually read. Otherwise, –1 is returned and *errno* is set to indicate the error.

SEE ALSO
`creat(), open(), stdio().`

NAME

scanf, fscanf, sscanf—convert formatted input

SYNOPSIS

```
#include <stdio.h>

int scanf(format [ , pointer ] ... )
char *format;

int fscanf(stream, format [ , pointer ] ... )
FILE *stream;
char *format;

int sscanf(s, format [ , pointer ] ... )
char *s, *format;
```

DESCRIPTION

Scanf reads characters from the standard input stream *stdin. Fscanf* reads characters from the named input stream, *stream. Sscanf* reads characters from the character string *s*. Each function converts the input according to a control string *(format)* and stores the results according to a set of *pointer* arguments that indicate where the converted output should be stored.

Format, the control string, contains specifications that control the interpretation of input sequences. The control string consists of characters to be matched in the input stream and/or conversion specifications that start with %. The control string may contain:

- White-space characters (blanks and tabs) that cause input to be read up to the next non-white-space character, except as described below.

- A character (any except %) that must match the next character of the input stream. To match a % character in the input stream, use "%%".

- Conversion specifications beginning with the character % and followed by an optional assignment suppression character *, an optional numeric maximum field width, an optional **l**, **m**, **n**, or **h** indicating the size of the receiving variable, and a conversion code.

An input field is defined relative to its conversion specification. The input field ends when the first character inappropriate for conversion is encountered or when the specified field width is exhausted. After conversion, the input pointer points to the inappropriate character.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, which is a pointer to a basic C type such as int or float.

Assignment can be suppressed by preceding a format character with a *. Assignment suppression causes an input field to be skipped; the field is read and converted but not assigned. Therefore *pointer* should be omitted when assignment of the corresponding input field is suppressed.

The format character dictates the interpretation of the input field. The following format characters are legal in a conversion specification, after %:

%	A single % is expected in the input at this point; no assignment is done.

> *The conversion characters* ***d, u, o,*** *and* ***x*** *may be preceded by* ***l*** *or* ***h*** *to indicate that a pointer to long or short, rather than int, is in the argument list. The* ***l*** *is ignored in this implementation because ints and longints are both 32 bits.*

d	A decimal integer is expected; the corresponding argument should be an integer pointer.
u	An unsigned decimal integer is expected; the corresponding argument should be an unsigned integer pointer.
o	An octal integer is expected; the corresponding argument should be an integer pointer.
x	A hexadecimal integer is expected; the corresponding argument should be an integer pointer.

> *The conversion characters* ***e, f,*** *and* ***g*** *may be preceded by* ***l, m,*** *or* ***n*** *to indicate that a pointer to double, comp, or extended, rather than float, is in the argument list.*

e,f,g	A floating-point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a float, double, comp, or extended, depending on the size specification. The input format for floating-point numbers is an optionally signed string of digits, possibly containing a decimal point, followed by an optional exponent field consisting of "E" or "e" followed by an optionally signed integer. In addition, infinity is represented by the string "INF", and NaNs are represented by the string "NAN", optionally followed by parentheses which may contain a string of digits (the NaN code). Case is ignored in the infinity and NaN strings.
s	A character string is expected; the corresponding argument should be a character pointer to an array of characters large enough to accept the string; a terminating null character (\0) is added automatically. The input field is terminated by a white-space (blank or tab) character.
c	A character is expected; the corresponding argument should be a character pointer. The normal skip over white space is suppressed in this case; to read the next non-white-space character, use "%1s". If a field width is given, the corresponding argument should refer to a character array; the indicated number of characters is read.
[	The left bracket introduces a scanset format. The input field is the maximal sequence of input characters consisting entirely of characters in

the scanset. When reading the input field, string data and the normal skip over leading white space are suppressed. The corresponding pointer argument must point to a character array large enough to hold the input field and the terminating null character (\0), which will be added automatically. The left bracket is followed by a set of characters (the scanset) and a terminating right bracket.

The circumflex (^), when it appears as the first character in the scanset, serves as a complement operator and redefines the scanset as the set of all characters <u>not</u> contained in the remainder of the scanset string.

The right bracket (]) ends the scanset. To include the right bracket as an element of the scanset, it must appear as the first character (possibly preceded by a circumflex) of the scanset; otherwise it will be interpreted syntactically as the closing bracket.

A range of characters may be represented by the construct *first-last*; thus the scanset [0123456789] may be expressed [0-9]. To use this convention, *first* must be less than or equal to *last* in the ASCII collating sequence; otherwise the minus (–) will stand for itself in the scanset. The minus will also stand for itself whenever it is the first or the last character in the scanset.

Scanf conversion terminates at EOF, at the end of the control string, or when an input character doesn't match the control string. In the latter case, the unmatched character is left unread in the input stream.

Scanf returns the number of successfully matched and assigned input items; this number can be zero when an early mismatch between an input character and the control string occurs. If the input ends before the first mismatch or conversion, EOF is returned.

EXAMPLES

Example 1. The call

```
int i; float x; char name[50];
scanf("%d%f%s", &i, &x, name);
```

with input

```
25 54.32E-1 hartwell
```

will assign the value 25 to *i*, and the value 5.432 to *x*; *name* will contain "hartwell\0".

Example 2. The call

```
int i; extended x; char name[50];
scanf("%2d%nf%*d %[0-9]", &i, &x, name);
```

with input

```
56789 0123 56a72
```

will assign 56 to *i*, 789.0 to *x*, skip 0123, and place the string "56\0" in *name*. The next call to getchar() will return "a".

Example 3. The call

```
int i;
scanf("answer1=%d", &i);
```

with input

```
answer1=51 answer2=45
```

will assign the value 51 to *i* since "answer1" is matched explicitly in the input stream; the input pointer will be left at the space before "answer2".

DIAGNOSTICS

These functions return EOF on end of input and a short count for missing or illegal data items.

NOTE

Trailing white space is left unread unless matched in the control string.

The success of literal matches and suppressed assignments is not directly determinable.

SEE ALSO

atof(), dec2num(), getc(), printf(), stdio(), str2dec(), strtol().
Apple Numerics Manual.

NAME

setbuf—assign buffering to a stream

SYNOPSIS

```
#include <stdio.h>

void setbuf (stream, buf)
FILE *stream;
char *buf;
```

DESCRIPTION

Setbuf is used after a stream has been opened but before it is read or written. *Setbuf* causes the character array pointed to by *buf* to be used instead of an automatically allocated buffer. If *buf* is a null character pointer, input/output will be completely unbuffered.

BUFSIZ, a constant defined in the <stdio.h> header file, indicates how big an array is needed:

```
      char buf[BUFSIZ];
```

A buffer is normally obtained from malloc() at the time of the first getc() or putc() on the file, except that the standard error stream *stderr* is normally not buffered.

Output streams directed to windows are either line-buffered or unbuffered.

NOTE

A common error is to allocate buffer space as an "automatic" variable in a code block and then fail to close the stream in the same block.

SEE ALSO

`fopen(), getc(), malloc(), putc().`

NAME

sinh, cosh, tanh—hyperbolic functions

SYNOPSIS

```
#include <math.h>

extended sinh (x)
extended x;

extended cosh (x)
extended x;

extended tanh (x)
extended x;
```

DESCRIPTION

Sinh, cosh, and *tanh* return, respectively, the hyberbolic sine, cosine, and tangent of their argument.

DIAGNOSTICS

Sinh, cosh, and *tanh* honor the floating-point exception flags—invalid operation, underflow, overflow, divide by zero, and inexact—as prescribed by the Standard Apple Numeric Environment (SANE).

SEE ALSO

Apple Numerics Manual.

NAME

stdio—standard buffered input/output package

SYNOPSIS

```
#include <stdio.h>

FILE *stdin, *stdout, *stderr;
```

DESCRIPTION

The standard I/O package constitutes an efficient user-level I/O buffering scheme. The inline macros getc() and putc() handle characters quickly. The macros getchar() and putchar(), and the higher-level routines fgetc(), fgets(), fprintf(), fputc(), fputs(), fread(), fscanf(), fwrite(), gets(), getw(), printf(), puts(), putw(), and scanf() all use getc() and putc(); they can be freely intermixed.

Any program that uses the standard I/O package must include the header file of pertinent macro definitions. The functions and constants mentioned in the standard I/O package are declared in the header file and need no further declaration. The header file is included as follows:

```
#include <stdio.h>
```

A file with associated buffering is called a *stream* and is declared to be a pointer to a defined type FILE. Fopen() creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. Normally, there are three open streams with constant pointers declared in the <stdio.h> header file and associated with the standard open files:

stdin	(standard input file)
stdout	(standard output file)
stderr	(standard error file)

A constant NULL (0) designates a nonexistent pointer.

An integer constant EOF (–1) is returned upon end-of-file or error by most integer functions that deal with streams. See the descriptions of the individual functions for details.

The constants and the following functions are implemented as macros: getc(), getchar(), putc(), putchar(), feof(), ferror(), clearerr(), and fileno(). Redeclaration of these names should be avoided.

NOTE

<Stdio.h> includes definitions other than those described above, but their use is not recommended.

DIAGNOSTICS

Invalid stream pointers cause serious errors, possibly including program termination. Individual function descriptions describe the possible error conditions.

SEE ALSO

open(), close(), lseek(), read(), write(), fclose(), ferror(), fopen(), fread(), fseek(), getc(), gets(), printf(), putc(), puts(), scanf(), setbuf(), ungetc().

NAME

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen,strchr, strrchr
—string operations

SYNOPSIS

```
char *strcat (s1, s2)
char *s1, *s2;

char *strncat (s1, s2, n)
char *s1, *s2;
int n;

int strcmp (s1, s2)
char *s1, *s2;

int strncmp (s1, s2, n)
char *s1, *s2;
int n;

char *strcpy (s1, s2)
char *s1, *s2;

char *strncpy (s1, s2, n)
char *s1, *s2;
int n;

int strlen (s)
char *s;

char *strchr (s, c)
char *s, c;

char *strrchr (s, c)
char *s, c;
```

DESCRIPTION

The arguments *s1*, *s2*, and *s* point to strings (arrays of characters terminated by a null character). The functions *strcat*, *strncat*, *strcpy*, and *strncpy* all alter *s1*. These functions do not check for overflow of the array pointed to by *s1*.

Strcat appends a copy of string *s2* to the end of string *s1*. *Strncat* appends at most *n* characters. Each function returns a pointer to the null-terminated result.

Strcmp performs a comparison of its arguments according to the ASCII collating sequence and returns an integer less than, equal to, or greater than 0 when *s1* is less than, equal to, or greater than *s2*, respectively. *Strncmp* makes the same comparison but looks at a maximum of *n* characters.

Strcpy copies string *s2* to string *s1*, stopping after the null character has been copied. *Strncpy* copies exactly *n* characters, truncating *s2* or adding null characters to *s1* if necessary. The result is not null-terminated if the length of *s2* is *n* or more. Each function returns *s1*.

Strlen returns the number of characters in *s*, not including the terminating null character.

Strchr (*strrchr*) returns a pointer to the first (last) occurrence of character *c* in string *s*; it returns a null pointer if *c* does not occur in the string. The null character terminating a string is considered to be part of the string. In previous versions of the Standard C Library, *strchr* was known as index() and *strrchr* was known as rindex().

WARNING

Overlapping moves may yield unexpected results.

NAME

sin, cos, tan, asin, acos, atan, atan2
—trigonometric functions

SYNOPSIS

```
#include <math.h>

extended sin (x)
extended x;

extended cos (x)
extended x;

extended tan (x)
extended x;

extended asin (x)
extended x;

extended acos (x)
extended x;

extended atan (x)
extended x;

extended atan2 (y, x)
extended x, y;
```

DESCRIPTION

Sin, cos, and *tan* return, respectively, the sine, cosine, and tangent of their argument, which is in radians.

Asin returns the arcsine of x, in the range $-\pi/2$ to $\pi/2$.

Acos returns the arccosine of x, in the range 0 to π.

Atan returns the arctangent of x, in the range $-\pi/2$ to $\pi/2$.

Atan2 returns the arctangent of y/x, in the range $-\pi$ to π, using the signs of both arguments to determine the quadrant of the return value.

For special cases, these functions return a NaN or infinity as appropriate.

DIAGNOSTICS

These functions honor the floating-point exception flags—invalid operation, underflow, overflow, divide by zero, and inexact—as prescribed by the Standard Apple Numeric Environment (SANE).

NOTE

Sin, cos, and *tan* have periods based on the nearest extended-precision representation of mathematical π. Hence these functions diverge from their mathematical counterparts as their argument becomes far from zero.

SEE ALSO

Apple Numerics Manual.

NAME
ungetc—push character back into input stream

SYNOPSIS
```
#include <stdio.h>

int ungetc (c, stream)
char c;
FILE *stream;
```

DESCRIPTION
Ungetc inserts the character *c* into the buffer associated with an input stream. That character, *c*, will be returned by the next getc() call on that stream. *Ungetc* returns *c* and leaves the file stream unchanged.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered.

If *c* equals EOF, *ungetc* does nothing to the buffer and returns EOF.

Fseek() erases all memory of inserted characters.

DIAGNOSTICS
For *ungetc* to perform correctly, a read must have been performed prior to the call of the *ungetc* function. *Ungetc* returns EOF if it can't insert the character. If *stream* is *stdin*, *ungetc* allows exactly one character to be pushed back onto the buffer without a previous read statement.

SEE ALSO
`fseek(), getc(), setbuf().`

NAME

unlink—remove a file

SYNOPSIS

```
int unlink (path)
char *path;
```

DESCRIPTION

Unlink deletes the file named by the pathname pointed to by *path.*

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

NAME

write—write on a file

SYNOPSIS

```
int write (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;
```

DESCRIPTION

Fildes is a file descriptor obtained from a creat() or open() call.

Write attempts to write *nbyte* bytes from the buffer pointed to by *buf* to the file associated with the *fildes*. Internal limitations may cause *write* to write fewer bytes than requested; the number of bytes actually written is indicated by the return value. Several calls to *write* may therefore be necessary to write out the contents of *buf*.

On devices capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file pointer. Upon return from *write*, the file pointer is incremented by the number of bytes actually written.

On nonseeking devices, writing always starts at the current position. The value of a file pointer associated with such a device is undefined.

If the O_APPEND file status flag is set—see open()—the file pointer is set to end-of-file prior to each write.

Write fails and the file pointer remains unchanged if *fildes* is not a valid file descriptor open for writing. **[EBADF]**

If you try to write more bytes than there is room for on the device, *write* writes as many bytes as possible. For example, if *nbyte* is 512 and there is room for 20 bytes more on the device, *write* writes 20 bytes and returns a value of 20. The next attempt to write a nonzero number of bytes will signal an error. **[ENOSPC]**

RETURN VALUE

Upon successful completion the number of bytes actually written is returned. Otherwise, −1 is returned and *errno* is set to indicate the error.

SEE ALSO

`creat(), lseek(), open().`

7. Macintosh Interface Libraries

This section contains the C definition of the Macintosh Interface Libraries. For complete documentation of these libraries, see *Inside Macintosh.*

This section is arranged alphabetically by library name. The Library Index, Section 8, indexes the defines, types, enumeration literals, global variables, and functions defined here and in the Standard C Library.

NAME

C interface to the Macintosh libraries

SYNOPSIS

```
#include <types.h>          /* common defines and types     */
#include <strings.h>        /* string conversions           */
#include <resources.h>      /* Resource Manager             */
#include <quickdraw.h>      /* QuickDraw                    */
#include <fonts.h>          /* Font Manager                 */
#include <events.h>         /* Event Manager                */
#include <windows.h>        /* Window Manager               */
#include <controls.h>       /* Control Manager              */
#include <menus.h>          /* Menu Manager                 */
#include <textedit.h>       /* TextEdit                    */
#include <dialogs.h>        /* Dialog Manager               */
#include <desk.h>           /* Desk Manager                 */
#include <scrap.h>          /* Scrap Manager                */
#include <toolutils.h>      /* Toolbox Utilities            */
#include <printing.h>       /* Printing Manager             */
#include <osutils.h>        /* Operating System Utilities   */
#include <memory.h>         /* Memory Manager               */
#include <segload.h>        /* Segment Loader               */
#include <osevents.h>       /* OS Event Manager             */
#include <files.h>          /* File Manager                 */
#include <devices.h>        /* Device Manager               */
#include <disks.h>          /* Disk Driver                  */
#include <sound.h>          /* Sound Driver                 */
#include <retrace.h>        /* Vertical Retrace Manager     */
#include <serial.h>         /* Serial Drivers               */
#include <packages.h>       /* packages                     */
#include <error.h>          /* System Error Handler         */
#include <sane.h>           /* SANE Numerics                */
```

DESCRIPTION

The C Interface provides C programs with access to all of the libraries defined in *Inside Macintosh.* Constants, types, and library routines are provided. The list of libraries appears above.

Header Files—Include the ".h" files in C programs to declare the defines, types, and functions provided by these libraries. Each library definition lists the includes necessary for use of that library. Functions whose declarations can be inferred from calls have been omitted from the header files. List the includes in the order in which the libraries are listed above.

Object File—The interface code is contained in file interface.obj. Link this file with the C program and other libraries. Not all functions require interface code. The linker includes interface code for only those routines that are called.

Interface Implementation—Most library routines are declared as external Pascal routines with trap numbers, and are trapped to directly by compiled code. Other routines are declared to be C routines and are called through interface glue.

Parameter Types—The C interfaces expect structures (including Points) to be passed by address. String parameters are C strings (i.e., zero-terminated) unless otherwise indicated. ResTypes and OSTypes can be expressed as character literals (e.g., 'MENU').

Spelling and Capitalization—The spelling and capitalization of identifiers is exactly as specified in *Inside Macintosh.* Constants, variables, parameter names, fields within structures, and enumeration-type elements begin with a lowercase letter. Routines and data types begin with an uppercase letter. Letters that begin new words in English are capitalized. All other letters are lowercase. When a name includes an acronym, the case of the entire acronym is determined by the case of the first letter (e.g., GetOSEvent, teJustLeft).

NAME

controls—Control Manager

SYNOPSIS

```
#include <types.h>
#include <controls.h>


/* Control Definition Procedures IDs */

#define pushButProc     0
#define checkBoxProc    1
#define radioButProc    2
#define useWFont        8
#define scrollBarProc  16

/* FindControl Result Codes */

#define inButton       10
#define inCheckbox     11
#define inUpButton     20
#define inDownButton   21
#define inPageUp       22
#define inPageDown     23
#define inThumb       129

/* DragControl Axis Constraints */

#define noConstraint    0
#define hAxisOnly       1
#define vAxisOnly       2

/* Messages to Control definition function */

#define drawCntl        0
#define testCntl        1
#define calcCRgns       2
#define initCntl        3
#define dispCntl        4
#define posCntl         5
#define thumbCntl       6
#define dragCntl        7
#define autoTrack       8


typedef struct ControlRecord {
   struct ControlRecord **nextControl;
   struct GrafPort *contrlOwner;
   Rect           contrlRect;
   unsigned char  contrlVis;
   unsigned char  contrlHilite;
   short          contrlValue;
   short          contrlMin;
   short          contrlMax;
   ProcHandle     contrlDefProc;
   Handle         contrlData;
   ProcPtr        contrlAction;
   long          contrlrfCon;
   Str255        contrlTitle;
} ControlRecord, *ControlPtr, **ControlHandle;


/* Initialization and Allocation */

ControlHandle NewControl(theWindow,boundsRect,title,visible,value,
      min,max,procID,refCon)
   struct GrafPort *theWindow;
   Rect *boundsRect;
   char *title;
   Boolean visible;
   short value;
   short min;
   short max;
   short procID;
   long refCon;
pascal ControlHandle GetNewControl(controlID,theWindow)
   short controlID;
   struct GrafPort *theWindow;
pascal void DisposeControl(theControl)
   ControlHandle theControl;
pascal void KillControls(theWindow)
   struct GrafPort *theWindow;

/* Control Display */

void SetCTitle(theControl,title)
   ControlHandle theControl;
   char *title;
void GetCTitle(theControl,title)
   ControlHandle theControl;
   char *title;
pascal void HideControl(theControl)
   ControlHandle theControl;
pascal void ShowControl(theControl)
   ControlHandle theControl;
pascal void DrawControls(theWindow)
   struct GrafPort *theWindow;
pascal void HiliteControl(theControl,hiliteState)
   ControlHandle theControl;
   short hiliteState;

/* Mouse Location */

short TestControl(theControl,thePoint)
   ControlHandle theControl;
   Point *thePoint;
short FindControl(thePoint,theWindow,theControl)
   Point *thePoint;
   struct GrafPort *theWindow;
   ControlHandle *theControl;
short TrackControl(theControl,startPt,actionProc)
   ControlHandle theControl;
   Point *startPt;
   ProcPtr actionProc;
```

```
/* Control Movement and Sizing */

pascal void MoveControl(theControl,h,v)
   ControlHandle theControl;
   short h,v;
void DragControl(theControl,startPt,limitRect,slopRect,axis)
   ControlHandle theControl;
   Point *startPt;
   Rect *limitRect;
   Rect *slopRect;
   short axis;
pascal void SizeControl(theControl,w,h)
   ControlHandle theControl;
   short w,h;

/* Control Settings and Range */

pascal void SetCtlValue(theControl,theValue)
   ControlHandle theControl;
   short theValue;
pascal short GetCtlValue(theControl)
   ControlHandle theControl;
pascal void SetCtlMin(theControl,minValue)
   ControlHandle theControl;
   short minValue;
pascal short GetCtlMin(theControl)
   ControlHandle theControl;
pascal void SetCtlMax(theControl,maxValue)
   ControlHandle theControl;
   short maxValue;
pascal short GetCtlMax(theControl)
   ControlHandle theControl;

/* Miscellaneous Utilities */

pascal void SetCRefCon(theControl,data)
   ControlHandle theControl;
   long data;
pascal long GetCRefCon(theControl)
   ControlHandle theControl;
pascal void SetCtlAction(theControl,actionProc)
   ControlHandle theControl;
   ProcPtr actionProc;
pascal ProcPtr GetCtlAction(theControl)
   ControlHandle theControl;
```

USER ROUTINES

```
pascal void MyAction()
pascal void MyAction(theControl,partCode)
   ControlHandle theControl;
   short partCode;

pascal long MyControl(varCode,theControl,message,param)
   short varCode;
   ControlHandle theControl;
```

```
    short message;
    long param;
```

DESCRIPTION

The Control Manager provides routines for creating and manipulating controls (e.g., buttons, scroll bars).

For more detailed information see the Control Manager chapter of *Inside Macintosh.*

NAME

desk—Desk Manager

SYNOPSIS

```
#include <types.h>
#include <desk.h>

/* Opening and Closing Desk Accessories */

short OpenDeskAcc(theAcc)
   char *theAcc;
pascal void CloseDeskAcc(refNum)
   short refNum;

/* Handling Events in Desk Accessories */

pascal void SystemClick(theEvent,theWindow)
   struct EventRecord *theEvent;
   struct GrafPort *theWindow;
pascal Boolean SystemEdit(editCmd)
   short editCmd;

/* Performing Periodic Actions */

pascal void SystemTask()

/* Advanced Routines */

pascal Boolean SystemEvent(theEvent)
   struct EventRecord *theEvent;
pascal void SystemMenu(menuResult)
   long menuResult;
```

DESCRIPTION

The Desk Manager supports desk accessories.

Warning: The names of desk accessories start with a null byte. The output parameter from GetMenuItem will return a string that begins with a null byte when a desk accessory is selected from the Apple menu. OpenDeskAcc skips over this null byte when interpreting its parameter.

For more detailed information see the Desk Manager chapter of *Inside Macintosh.*

NOTE

Desk accessories do not have an A5 global area. Therefore all of the code for a desk accessory must reside in a single sement; no global variables may be declared; and no string constants may be used.

NAME

devices—Device Manager

SYNOPSIS

```
#include <types.h>
#include <osutils.h>
#include <devices.h>


/* error codes */

#define  abortErr     (-27)    /* I/O request aborted by KillIO */
#define  badUnitErr   (-21)    /* RefNum doesn't match unit table */
#define  controlErr   (-17)    /* Driver can't respond to this call */
#define  dInstErr     (-26)    /* Couldn't find driver in resource file */
#define  dRemoveErr   (-25)    /* Tried to remove an open driver */
#define  noErr          0      /* No error */
#define  notOpenErr   (-28)    /* Driver isn't open */
#define  openErr      (-23)    /* Requested r/w permission doesn't */
                               /*    match driver's open permission */
#define  readErr      (-19)    /* Driver can't respond to Read calls */
#define  statusErr    (-18)    /* Driver can't respond to Status call */
#define  unitEmptyErr (-22)    /* Reference number specifies nil handle */
                               /*   in unit table */
#define  writErr      (-20)    /* Driver can't respond to Write calls */


typedef struct CntrlParam {
   struct QElem *qLink;        /* next queue entry */
   short       qType;          /* queue type */
   short       ioTrap;         /* routine trap */
   Ptr         ioCmdAddr;      /* routine address */
   ProcPtr     ioCompletion;   /* completion routine */
   OSErr       ioResult;       /* result code */
   char       *ioNamePtr;      /* volume or file name */
   short       ioVRefNum;      /* volume refnum or drive number */
   short       ioCRefNum;      /* refnum for I/O operations */
   short       csCode;         /* word for control status code */
   short       csParam[11]     /* operation-defined parameters */
} CntrlParam;


typedef struct DCtlEntry {
   Ptr         dCtlDriver;     /* ptr to ROM or handle to RAM driver */
   short       dCtlFlags;      /* flags */
   QHdr        dCtlQHdr;       /* driver's I/O queue */
   long        dCtlPosition;   /* byte pos used by read and write */
   Handle      dCtlStorage;    /* handle to RAM driver's storage */
   short       dCtlRefNum;     /* driver's reference number */
   long        dCtlCurTicks;   /* counter for timing system task calls */
   Ptr         dCtlWindow;     /* ptr to driver's window (if any) */
   short       dCtlDelay;      /* # of ticks between sysTask calls */
   short       dCtlEMask;      /* desk accessory event mask */
   short       dCtlMenu;       /* menu ID of menu associated w/ driver */
} DCtlEntry, *DCtlPtr, **DCtlHandle;
```

```
/* High-Level Routines */

OSErr OpenDriver(name,refNum)
   char *name;
   short *refNum;
OSErr CloseDriver(refNum)
   short refNum;
OSErr Control(refNum,csCode,csParam)
   short refNum,csCode;
   Ptr csParam;
OSErr Status(refNum,csCode,csParam)
   short refNum,csCode;
   Ptr csParam;
OSErr KillIO(refNum)
   short refNum;


/* Low-Level Routines */

OSErr PBControl(paramBlock,async)
   CntrlParam *paramBlock;
   Boolean async;
OSErr PBStatus(paramBlock,async)
   CntrlParam *paramBlock;
   Boolean async;
OSErr PBKillIO(paramBlock,async)
   CntrlParam *paramBlock;
   Boolean async;


/* Accessing a Driver's I/O Queue */

DCtlHandle GetDCtlEntry(refNum)
   short refNum;
```

DESCRIPTION

The Device Manager controls the exchange of information between an application and devices.

For more detailed information see the Device Manager chapter of *Inside Macintosh.*

NAME
dialogs—Dialog Manager

SYNOPSIS
```
#include <types.h>
#include <quickdraw.h>
#include <windows.h>
#include <dialogs.h>

/* Item types */

#define ctrlItem       4       /* add to following four constants       */
#define btnCtrl        0       /* standard button control               */
#define chkCtrl        1       /* standard check box control            */
#define radCtrl        2       /* standard "radio button" control       */
#define resCtrl        3       /* control defined in control template   */
#define statText       8       /* static text                           */
#define editText      16       /* editable text (dialog only)           */
#define iconItem      32       /* icon                                  */
#define picItem       64       /* Quickdraw picture                     */
#define userItem       0       /* application-defined item (dialog only)*/
#define itemDisable 128        /* add to any of above to disable        */

/* Item numbers of OK and Cancel buttons */

#define ok             1       /* OK button is first by convention      */
#define cancel         2       /* Cancel button is second by convention */

/* Resource IDs of Alert icons */

#define stopIcon       0
#define noteIcon       1
#define ctnIcon        2


typedef WindowPtr DialogPtr;
typedef struct DialogRecord {
   WindowRecord   window;
   Handle         items;
   struct TERec **textH;
   short          editField;
   short          editOpen;
   short          aDefItem;
} DialogRecord, *DialogPeek;

typedef struct DialogTemplate {
   Rect     boundsRect;
   short    procID;
   Boolean  visible;
   Boolean  filler1;
   Boolean  goAwayFlag;
   Boolean  filler2;
   long     refCon;
   short    itemsID;
   Str255   title;
} DialogTemplate, *DialogTPtr, **DialogTHndl;
```

```
typedef long StageList;

typedef struct AlertTemplate {
   Rect      boundsRect;
   short     itemsID;
   StageList stages;
} AlertTemplate, *AlertTPtr, **AlertTHndl;



/* Initialization */

pascal void InitDialogs(restartProc)
   ProcPtr restartProc;
pascal void ErrorSound(soundProc)
   ProcPtr soundProc;
void SetDAFont(fontNum)
   short fontNum;

/* Creating and Disposing of Dialogs */

DialogPtr NewDialog(dStorage,boundsRect,title,visible,procID,behind,
      goAwayFlag,refCon,items)
   Ptr dStorage;
   Rect *boundsRect;
   char *title;
   Boolean visible;
   short procID;
   WindowPtr behind;
   Boolean goAwayFlag;
   long refCon;
   Handle items;
pascal DialogPtr GetNewDialog(dialogID,dStorage,behind)
   short dialogID;
   Ptr dStorage;
   WindowPtr behind;
pascal void CloseDialog(theDialog)
   DialogPtr theDialog;
pascal void DisposDialog(theDialog)
   DialogPtr theDialog;
pascal void CouldDialog(dialogID)
   short dialogID;
pascal void FreeDialog(dialogID)
   short dialogID;

/* Handling Dialog Events */

pascal void ModalDialog(filterProc,itemHit)
   ProcPtr filterProc;
   short *itemHit;
pascal Boolean IsDialogEvent(theEvent)
   struct EventRecord *theEvent;
pascal Boolean DialogSelect(theEvent,theDialog,itemHit)
   struct EventRecord *theEvent;
   DialogPtr *theDialog;
   short *itemHit;
void DlgCut(theDialog)
```

```
   DialogPtr theDialog;
void DlgCopy(theDialog)
   DialogPtr theDialog;
void DlgPaste(theDialog)
   DialogPtr theDialog;
void DlgDelete(theDialog)
   DialogPtr theDialog;
pascal void DrawDialog(theDialog)
   DialogPtr theDialog;

/* Invoking Alerts */

pascal short Alert(alertID,filterProc)
   short alertID;
   ProcPtr filterProc;
pascal short StopAlert(alertID,filterProc)
   short alertID;
   ProcPtr filterProc;
pascal short NoteAlert(alertID,filterProc)
   short alertID;
   ProcPtr filterProc;
pascal short CautionAlert(alertID,filterProc)
   short alertID;
   ProcPtr filterProc;
pascal void CouldAlert(alertID)
   short alertID;
pascal void FreeAlert(alertID)
   short alertID;

/* Manipulating Items in Dialogs and Alerts */

void ParamText(param0,param1,param2,param3)
   char *param0;
   char *param1;
   char *param2;
   char *param3;
pascal void GetDItem(theDialog,itemNo,type,item,box)
   DialogPtr theDialog;
   short itemNo;
   short *type;
   Handle *item;
   Rect *box;
pascal void SetDItem(theDialog,itemNo,type,item,box)
   DialogPtr theDialog;
   short itemNo;
   short type;
   Handle item;
   Rect *box;
void GetIText(item,text)
   Handle item;
   char *text;
void SetIText(item,text)
   Handle item;
   char *text;
pascal void SelIText(theDialog,itemNo,startSel,endSel)
   DialogPtr theDialog;
   short itemNo;
```

```
      short startSel;
      short endSel;
   short GetAlrtStage()
   void ResetAlrtStage()
```

USER ROUTINES

```
   pascal void MyItem(theWindow,itemNo)
      struct GrafPort *theWindow;
      short itemNo;
   pascal void MySound(soundNo)
      short soundNo;
   pascal Boolean MyFilter(theDialog,theEvent,itemHit)
      DialogPtr theDialog;
      struct EventRecord *theEvent;
      short *itemHit;
```

DESCRIPTION

The Dialog Manager supports dialog boxes and the alert mechanism.

For more detailed information see the Dialog Manager chapter of *Inside Macintosh.*

NOTE

StageList is defined as a long, rather than specifying the bits.

NAME

disks—Disk Driver

SYNOPSIS

```
#include <types.h>
#include <disks.h>


/* error codes */

#define  firstDskErr  (-84)     /* First of the range of disk errors */
#define  lastDskErr   (-64)     /* Last of the range of disk errors */

#define  badBtSlpErr  (-70)     /* Bad address mark */
#define  badCksmErr   (-69)     /* Bad address mark */
#define  badDBtSlp    (-73)     /* Bad data mark */
#define  badDCksum    (-72)     /* Bad data mark */
#define  cantStepErr  (-75)     /* Hardware error */
#define  dataVerErr   (-68)     /* Read-verify failed */
#define  initIWMErr   (-77)     /* Hardware error */
#define  noAdrMkErr   (-67)     /* Can't find an address mark */
#define  noDriveErr   (-64)     /* Drive isn't connected */
#define  noDtaMkErr   (-71)     /* Can't find data mark */
#define  noErr           0      /* No error  */
#define  noNybErr     (-66)     /* Disk is probably blank */
#define  nsDrvErr     (-56)     /* No such drive */
#define  offLinErr    (-65)     /* No disk in drive */
#define  paramErr     (-50)     /* Bad positioning information */
#define  sectNFErr    (-81)     /* Can't find sector */
#define  seekErr      (-80)     /* Hardware error */
#define  spdAdjErr    (-79)     /* Hardware error */
#define  tk0BadErr    (-76)     /* Hardware error */
#define  twoSideErr   (-78)     /* Tried to read side 2, on 1 side drive */
#define  wPrErr       (-44)     /* Disk is locked */
#define  wrUnderrun   (-74)     /* write underrun occurred */

typedef struct DrvSts {
   short       track;           /* current track */
   char        writeProt;       /* bit 7=1 if volume is locked */
   char        diskInPlace;     /* disk in place */
   char        installed;       /* drive installed */
   char        sides;           /* bit 7=0 if single-sided drive */
   struct QElem *qLink;         /* next queue entry */
   short       qType;           /* not used */
   short       dQDrive;         /* drive number */
   short       dQRefNum;        /* driver reference number */
   short       dQFSID;          /* file-system identifier */
   char        twoSideFmt;      /* -1 if two-sided disk */
   char        needsFlush;      /* reserved */
   short       diskErrs;        /* error count */
} DrvSts;


OSErr DiskEject(drvNum)
   short drvNum;
OSErr SetTagBuffer(buffPtr)
   Ptr buffPtr;
```

```
OSErr DriveStatus(drvNum,status)
   short drvNum;
   DrvSts *status;
```

DESCRIPTION

The Disk Driver is a Macintosh device driver used for storing and retrieving information on Macintosh 3 1/2-inch disk drives.

For more detailed information see the Disk Driver chapter of *Inside Macintosh.*

NAME

error—System Error Handler

SYNOPSIS

```
#include <error.h>

/* error codes */

#define  dsBusErr        1    /* Bus Error                    */
#define  dsAddressErr    2    /* Address Error                */
#define  dsIllInstErr    3    /* Illegal Instruction          */
#define  dsZeroDivErr    4    /* Zero Divide                  */
#define  dsChkErr        5    /* Check Exception              */
#define  dsOvflowErr     6    /* TrapV Exception              */
#define  dsPrivErr       7    /* Privilege Violation          */
#define  dsTraceErr      8    /* Trace Exception              */
#define  dsLineAErr      9    /* Line 1010 Exception          */
#define  dsLineFErr     10    /* Line 1111 Exception          */
#define  dsMiscErr      11    /* Miscellaneous Exception      */
#define  dsCoreErr      12    /* Unimplemented Core Routine   */
#define  dsIrqErr       13    /* Spurious Interrupt           */
#define  dsIOCoreErr    14    /* I/O System Error             */
#define  dsLoadErr      15    /* Segment Loader Error         */
#define  dsFPErr        16    /* Floating Point Error         */
#define  dsNoPackErr    17    /* Can't Load Package 0         */
#define  dsNoPk1        18    /* Can't Load Package 1         */
#define  dsNoPk2        19    /* Can't Load Package 2         */
#define  dsNoPk3        20    /* Can't Load Package 3         */
#define  dsNoPk4        21    /* Can't Load Package 4         */
#define  dsNoPk5        22    /* Can't Load Package 5         */
#define  dsNoPk6        23    /* Can't Load Package 6         */
#define  dsNoPk7        24    /* Can't Load Package 7         */
#define  dsMemFullErr   25    /* Out of Memory                */
#define  dsBadLaunch    26    /* Segment Loader Error         */
#define  dsFSErr        27    /* File Map Trashed             */
#define  dsStkNHeap     28    /* Stack Overflow Error         */
#define  dsReinsert     30    /* Please Insert the Disk       */
#define  dsNotThe1      31    /* This is Not the Correct Disk */
#define  dsSysErr    32767    /* System Error                 */


void SysError(errorCode)
   short errorCode;
```

DESCRIPTION

The System Error Handler is the part of the Macintosh Operating System that assumes control when a fatal error occurs. For more detailed information see the System Error Handler chapter of *Inside Macintosh.*

NAME

events—Toolbox Event Manager

SYNOPSIS

```
#include <types.h>
#include <events.h>

/* Event Types */

#define  nullEvent         0
#define  mouseDown         1
#define  mouseUp           2
#define  keyDown           3
#define  keyUp             4
#define  autoKey           5
#define  updateEvt         6
#define  diskEvt           7
#define  activateEvt       8
#define  networkEvt       10
#define  driverEvt        11
#define  app1Evt          12
#define  app2Evt          13
#define  app3Evt          14
#define  app4Evt          15

/* Masks for accessing keyboard event message */

#define  charCodeMask  0x000000FF
#define  keyCodeMask   0x0000FF00

/* Event Masks */

#define  mDownMask         2
#define  mUpMask           4
#define  keyDownMask       8
#define  keyUpMask        16
#define  autoKeyMask      32
#define  updateMask       64
#define  diskMask        128
#define  activMask       256
#define  networkMask    1024
#define  driverMask     2048
#define  app1Mask       4096
#define  app2Mask       8192
#define  app3Mask      16384
#define  app4Mask    (-32768)
#define  everyEvent     (-1)

/* Modifiers */

#define  activeFlag        1
#define  btnState        128
#define  cmdKey          256
#define  shiftKey        512
#define  alphaLock      1024
#define  optionKey      2048
```

```
typedef struct EventRecord {
   short what;
   long  message;
   long  when;
   Point where;
   short modifiers;
} EventRecord;

typedef long KeyMap[4];


/* Accessing Events */

pascal Boolean GetNextEvent(eventMask,theEvent)
   short eventMask;
   EventRecord *theEvent;
pascal Boolean EventAvail(eventMask,theEvent)
   short eventMask;
   EventRecord *theEvent;


/* Reading the Mouse */

pascal void GetMouse(mouseLoc)
   Point *mouseLoc;
pascal Boolean Button()
pascal Boolean StillDown()
pascal Boolean WaitMouseUp()


/* Reading the Keyboard and Keypad */

pascal void GetKeys(theKeys)
   KeyMap *theKeys;


/* Miscellaneous Routines */

pascal long TickCount()
long GetDblTime()
long GetCaretTime()
```

DESCRIPTION

The Toolbox Event Manager provides access to the Macintosh keyboard, keypad, and mouse. The keyboard bit map returned by the function GetKeys() is organized as shown below. The bits aren't numbered as you might expect. Here is the actual numbering scheme, corresponding to the key code numbers given in the Key Codes figure in the Toolbox Event Manager chapter of *Inside Macintosh.*

```
keyMap[0]   keyMap[1]   keyMap[2]   keyMap[3]
       |            |          |           |
       |            |          |           |
       v            v          v           v
 bytes b0,b1,b2,b3  b4,b5, ....        ....b12,b13,b14,b15
       |  \                                             |
       |   \____________                                |
       v                \                               v
 bits  7,6,5,4,3,2,1,0   |     127,126,125,124,123,122,121,120
                         v
               15,14,13,12,11,10,9,8
```

NAME

files—File Manager

SYNOPSIS

```
#include <types.h>
#include <files.h>


/* error codes */

#define  badMDBErr     (-60)    /* Master directory block is bad, reinit */
#define  bdNamErr      (-37)    /* Bad file or volume name (zero length?)*/
#define  dirFulErr     (-33)    /* File directory full */
#define  dskFulErr     (-34)    /* All allocation blocks are full */
#define  dupFNErr      (-48)    /* File by that name already exists */
#define  eofErr        (-39)    /* Logical EOF reached during read */
#define  extFSErr      (-58)    /* External file system; identifier is */
                                /*    non-zero or refNum is > 1024     */
#define  fBsyErr       (-47)    /* One or more files are open */
#define  fLckdErr      (-45)    /* File Locked */
#define  fnfErr        (-43)    /* File not found */
#define  fnOpnErr      (-38)    /* File not open */
#define  fsRnErr       (-59)    /* Problem during rename */
#define  ioErr         (-36)    /* Disk I/O error */
#define  noErr           0      /* No error */
#define  nsDrvErr      (-56)    /* No such drive in the drive queue */
#define  noMacDskErr   (-57)    /* Volume lacks Mac-format directory */
#define  nsvErr        (-35)    /* Specified volume doesn't exist */
#define  opWrErr       (-49)    /* Only one writer allowed */
#define  paramErr      (-50)    /* Parameter's don't specify an existing  */
                                /*   volume and there's no default volume */
#define  permErr       (-54)    /* Permission doesn't allow writing */
#define  posErr        (-40)    /* Attempted to position before start */
#define  rfNumErr      (-51)    /* Bad reference number */
#define  tmfoErr       (-42)    /* Only 12 files can be open at once */
#define  volOffLinErr  (-53)    /* Volume not on-line */
#define  volOnLinErr   (-55)    /* Volume is already mounted and on-line */
#define  vLckdErr      (-46)    /* Volume is locked by a software flag */
#define  wrPermErr     (-61)    /* Permission does not allow writing */
#define  wPrErr        (-44)    /* Volume is locked by a hardware setting */


/* Flags in file information used by the Finder */

#define  fHasBundle   8192      /* set if file has a bundle */
#define  fInvisible  16384      /* set if file's icon is invisible */
#define  fTrash         (-3)    /* file is in trash window */
#define  fDesktop       (-2)    /* file in on desktop */
#define  fDisk           0      /* file is in disk window */


/* Values for posMode and ioPosMode */

#define  fsAtMark        0      /* at current position of mark */
                                /*  (posOff or ioPosOffset ignored) */
#define  fsFromStart     1      /* offset relative to beginning of file */
#define  fsFromLEOF      2      /* offset relative to logical EOF */
```

```
#define  fsFromMark      3     /* offset relative to current mark */
#define  rdVerify       64     /* added to above for read-verify */


/* Values for requesting read/write access */

#define  fsCurPerm       0     /* whatever is currently allowed */
#define  fsRdPerm        1     /* request to read only */
#define  fsWrPerm        2     /* request to write only */
#define  fsRdWrPerm      3     /* request to read and write */


typedef struct FInfo {
   OSType     fdType;          /* the type of the file */
   OSType     fdCreator;       /* file's creator */
   short      fdFlags;         /* flags.  hasBundle, invisible, etc. */
   Point      fdLocation;      /* file's location in window */
   short      fdFldr;          /* folder containing file */
} FInfo;


/* Parameter Blocks */

typedef struct IOParam {
   struct QElem *qLink;        /* next queue entry */
   short      qType;           /* queue type */
   short      ioTrap;          /* routine trap */
   Ptr        ioCmdAddr;       /* routine address */
   ProcPtr    ioCompletion;    /* completion routine */
   OSErr      ioResult;        /* result code */
   StringPtr  ioNamePtr;       /* volume of file name */
   short      ioVRefNum;       /* volume refnum or drive number */
   short      ioRefNum;        /* path reference number */
   char       ioVersNum;       /* version number */
   char       ioPermssn;       /* read/write permission */
   Ptr        ioMisc;          /* miscellaneous (Note: use high byte  */
                               /*    for PBSetFVers) */
   Ptr        ioBuffer;        /* data buffer */
   long       ioReqCount;      /* requested number of bytes */
   long       ioActCount;      /* actual byte count completed */
   short      ioPosMode;       /* newline char and type of positioning */
   long       ioPosOffset;     /* size of positioning offset */
} IOParam;


typedef struct FileParam {
   struct QElem *qLink;        /* next queue entry */
   short      qType;           /* queue type */
   short      ioTrap;          /* routine trap */
   Ptr        ioCmdAddr;       /* routine address */
   ProcPtr    ioCompletion;    /* completion routine */
   OSErr      ioResult;        /* result code */
   StringPtr  ioNamePtr;       /* volume of file name */
   short      ioVRefNum;       /* volume refnum or drive number */
   short      ioFRefNum;       /* path reference number */
   char       ioFVersNum;      /* version number */
   char       filler1;         /* not used */
   short      ioFDirIndex;      /* sequence number of file */
   char       ioFlAttrib;       /* file attributes */
   char       ioFlVersNum;      /* version number */
   FInfo      ioFlFndrInfo;     /* information used by the Finder */
   long       ioFlNum;          /* file number */
   short      ioFlStBlk;        /* first block of data fork */
   long       ioFlLgLen;        /* logical EOF of data fork */
   long       ioFlPyLen;        /* phyisical EOF of data fork */
   short      ioFlRStBlk;       /* first block of resource fork */
   long       ioFlRLgLen;       /* logical EOF of resource fork */
   long       ioFlRPyLen;       /* physical EOF of resource fork */
   long       ioFlCrDat;        /* date and time of creation */
   long       ioFlMdDat;        /* date and time of last modification */
} FileParam;


typedef struct VolumeParam {
   struct QElem *qLink;         /* next queue entry */
   short      qType;            /* queue type */
   short      ioTrap;           /* routine trap */
   Ptr        ioCmdAddr;        /* routine address */
   ProcPtr    ioCompletion;     /* completion routine */
   OSErr      ioResult;         /* result code */
   StringPtr  ioNamePtr;        /* volume of file name */
   short      ioVRefNum;        /* volume refnum or drive number */
   long       filler2;          /* not used */
   short      ioVolIndex;       /* volume index */
   long       ioVCrDate;        /* date and time of initialization */
   long       ioVLsBkUp;        /* date and time of last volume backup */
   short      ioVAtrb;          /* bit 15=1 if volume locked */
   short      ioVNmFls;         /* number of files in file directory */
   short      ioVDirSt;         /* first block of file directory */
   short      ioVBlLn;          /* number of blocks in file directory */
   short      ioVNmAlBlks;      /* number of alloc blocks on volume */
   long       ioVAlBlkSiz;      /* number of bytes per alloc block */
   long       ioVClpSiz;        /* number of bytes to Allocate */
   short      ioAlBlSt;         /* first block in volume block map */
   long       ioVNxtFNum;       /* next free file number */
   short      ioVFrBlk;         /* number of free allocation blocks */
} VolumeParam;


typedef struct VCB {
   struct QElem *qLink;         /* next queue entry */
   short      qType;            /* not used */
   short      vcbFlags;         /* bit 15=1 if dirty */
   short      vcbSigWord;       /* always 0xd2d7 */
   long       vcbCrDate;        /* date volume was initialized */
   long       vcbLsBkUp;        /* date of last backup */
   short      vcbAtrb;          /* volume attributes */
   short      vcbNmFls;         /* number of files in directory */
   short      vcbDirSt;         /* directory's first block */
   short      vcbBlLn;          /* length of file directory */
   short      vcbNmBlks;        /* number of allocation blocks */
   long       vcbAlBlkSiz;      /* size of allocation blocks */
   long       vcbClpSiz;        /* number of bytes to Allocate */
   short      vcbAlBlSt;        /* first block in block map */
   long        vcbNxtFNum;       /* next unused file number */
   short       vcbFreeBks;       /* number of unused blocks */
   String(27) vcbVN;             /* volume name */
   short       vcbDrvNum;        /* drive number */
   short       vcbDRefNum;       /* driver reference number */
   short       vcbFSID;          /* file system identifier */
   short       vcbVRefNum;       /* volume reference number */
   Ptr         vcbMAdr;          /* location of block map */
   Ptr         vcbBufAdr;        /* location of volume buffer */
   short       vcbMLen;          /* number of bytes in block map */
   short       vcbDirIndex;      /* used internally */
   short       vcbDirBlk;        /* used internally */
} VCB;


typedef struct DrvQEl {
   struct QElem *qLink;          /* next queue entry */
   short       qType;            /* not used */
   short       dQDrive;          /* drive number */
   short       dQRefNum;         /* drive reference number */
   short       dQFSID;           /* file system identifier */
   short       dQDrvSize;        /* number of logical blocks */
} DrvQEl, *DrvQElPtr;


/* --- High-Level Routines ------------------------------------------ */


/* Accessing Volumes */

OSErr GetVInfo(drvNum,volName,vRefNum,freeBytes)
   short drvNum;
   char *volName;
   short *vRefNum;
   long *freeBytes;
OSErr GetVol(volName,vRefNum)
   char *volName;
   short *vRefNum;
OSErr SetVol(volName,vRefNum)
   char *volName;
   short vRefNum;
OSErr FlushVol(volName,vRefNum)
   char *volName;
   short vRefNum;
OSErr UnmountVol(volName,vRefNum)
   char *volName;
   short vRefNum;
OSErr Eject(volName,vRefNum)
   char *volName;
   short vRefNum;


/* Changing File Contents */

OSErr Create(fileName,vRefNum,creator,fileType)
   char *fileName;
   short vRefNum;
   OSType creator,fileType;
OSErr FSOpen(fileName,vRefNum,refNum)
   char *fileName;
   short vRefNum,*refNum;
OSErr OpenRF(fileName,vRefNum,refNum)
   char *fileName;
   short vRefNum,*refNum;
OSErr FSRead(refNum,count,buffPtr)
   short refNum;
   long *count;
   Ptr buffPtr;
OSErr FSWrite(refNum,count,buffPtr)
   short refNum;
   long *count;
   Ptr buffPtr;
OSErr GetFPos(refNum,filePos)
   short refNum;
   long *filePos;
OSErr SetFPos(refNum,posMode,posOff)
   short refNum,posMode;
   long posOff;
OSErr GetEOF(refNum,logEOF)
   short refNum;
   long *logEOF;
OSErr SetEOF(refNum,logEOF)
   short refNum;
   long logEOF;
OSErr Allocate(refNum,count)
   short refNum;
   long *count;
OSErr GetVRefNum(pathRefNum,vRefNum)
   short pathRefNum;
   short *vRefNum;
OSErr FSClose(refNum)
   short refNum;


/* Changing Information About Files */

OSErr GetFInfo(fileName,vRefNum,fndrInfo)
   char *fileName;
   short vRefNum;
   FInfo *fndrInfo;
OSErr SetFInfo(fileName,vRefNum,fndrInfo)
   char *fileName;
   short vRefNum;
   FInfo *fndrInfo;
OSErr SetFLock(fileName,vRefNum)
   char *fileName;
   short vRefNum;
OSErr RstFLock(fileName,vRefNum)
   char *fileName;
   short vRefNum;
OSErr Rename(oldName,vRefNum,newName)
   char *oldName;
   short vRefNum;
   char *newName;
```

```
OSErr FSDelete(fileName,vRefNum)
   char *fileName;
   short vRefNum;


/* --- Low-Level Routines ------------------------------------------ */


void FInitQueue()
void AddDrive(drvrRefNum,drvNum,qEl);
   short drvrRefNum,drvNum;
   drvQElPtr qEL;


/* Accessing Volumes */

OSErr PBMountVol(paramBlock)
   VolumeParam *paramBlock;
OSErr PBGetVInfo(paramBlock,async)
   VolumeParam *paramBlock;
   Boolean async;
OSErr PBGetVol(paramBlock,async)
   VolumeParam *paramBlock;
   Boolean async;
OSErr PBSetVol(paramBlock,async)
   VolumeParam *paramBlock;
   Boolean async;
OSErr PBFlushVol(paramBlock,async)
   VolumeParam *paramBlock;
   Boolean async;
OSErr PBUnmountVol(paramBlock)
   VolumeParam *paramBlock;
OSErr PBOffLine(paramBlock)
   VolumeParam *paramBlock;
OSErr PBEject(paramBlock)
   VolumeParam *paramBlock;


/* Changing File Contents */

OSErr PBCreate(paramBlock,async)
   FileParam *paramBlock;
   Boolean async;
OSErr PBOpen(paramBlock,async)
   IOParam *paramBlock;
   Boolean async;
OSErr PBOpenRF(paramBlock,async)
   IOParam *paramBlock;
   Boolean async;
OSErr PBRead(paramBlock,async)
   IOParam *paramBlock;
   Boolean async;
OSErr PBWrite(paramBlock,async)
   IOParam *paramBlock;
   Boolean async;
OSErr PBGetFPos(paramBlock,async)
   IOParam *paramBlock;
   Boolean async;
OSErr PBSetFPos(paramBlock,async)
   IOParam *paramBlock;
   Boolean async;
OSErr PBGetEOF(paramBlock,async)
   IOParam *paramBlock;
   Boolean async;
OSErr PBSetEOF(paramBlock,async)
   IOParam *paramBlock;
   Boolean async;
OSErr PBAllocate(paramBlock,async)
   IOParam *paramBlock;
   Boolean async;
OSErr PBFlushFile(paramBlock,async)
   IOParam *paramBlock;
   Boolean async;
OSErr PBClose(paramBlock,async)
   IOParam *paramBlock;
   Boolean async;


/* Changing Information About Files */

OSErr PBGetFInfo(paramBlock,async)
   FileParam *paramBlock;
   Boolean async;
OSErr PBSetFInfo(paramBlock,async)
   FileParam *paramBlock;
   Boolean async;
OSErr PBSetFLock(paramBlock,async)
   FileParam *paramBlock;
   Boolean async;
OSErr PBRstFLock(paramBlock,async)
   FileParam *paramBlock;
   Boolean async;
OSErr PBSetFVers(paramBlock,async)
   IOParam *paramBlock;
   Boolean async;
OSErr PBRename(paramBlock,async)
   IOParam *paramBlock;
   Boolean async;
OSErr PBDelete(paramBlock,async)
   FileParam *paramBlock;
   Boolean async;


/* Accessing Queues */

struct QHdr *GetFSQHdr()
struct QHdr *GetVCBQHdr()
struct QHdr *GetDrvQHdr()
```

DESCRIPTION

The File Manager controls the exchange of information between an application and files.

Warning: The low-level routines that use strings take as input and return as output pointers to Pascal-style strings (string length in first byte). However, the high-level routines use C-style strings (terminated by a null character) as input and output parameters.

For more detailed information see the File Manager chapter of *Inside Macintosh.*

NOTE

An I/O completion routine cannot reliably access any globals, strings, or other functions outside its segment.

NAME

fonts—Font Manager

SYNOPSIS

```
#include <types.h>
#include <fonts.h>

#define  systemFont  0
#define  applFont    1
#define  newYork     2
#define  geneva      3
#define  monaco      4
#define  venice      5
#define  london      6
#define  athens      7
#define  sanFran     8
#define  toronto     9
#define  cairo       11
#define  losAngeles  12
#define  times       20
#define  helvetica   21
#define  courier     22
#define  symbol      23
#define  taliesin    24
#define  kanji       25

#define commandMark   '\021'
#define checkMark     '\022'
#define diamondMark   '\023'
#define appleMark     '\024'

#define  propFont    0x9000
#define  fixedFont   0xB000
#define  fontWid     0xACB0


typedef struct FMInput {
   short         family;         /* e.g. New York  */
   short         size;           /* e.g. 12 Point */
   Style         face;           /* e.g. bold | underline   */
   Boolean       needBits;       /* bits or just measurement */
   short         device;         /* always 0 for display */
   Point         numer;          /* current drawing scale */
   Point         denom;          /* current drawing scale */
} FMInput;

typedef struct FMOutput {
   short         errNum;         /* not used */
   Handle        fontHandle;     /* Handle to font */
   unsigned char boldPixels;     /* pixels of horizontal smear */
   unsigned char italicPixels;   /* pixels of horizontal shear */
   unsigned char ulOffset;       /* pixels below baseline */
   unsigned char ulShadow;       /* pixels in halo */
   unsigned char ulThick;        /* thickness of underline */
   unsigned char shadowPixels;   /* pixels to shadow (0 .. 3) */
   char          extra;          /* extra white pixels/char */
   unsigned char ascent;         /* ascent */
   unsigned char descent;        /* descent */
   unsigned char widMax;         /* maximum character width */
   char          leading;        /* leading between lines */
   char          unused;         /* not used */
   Point         numer;          /* current drawing scale */
   Point         denom;          /* current drawing scale */
} FMOutput, *FMOutPtr;

typedef struct FontRec {
   short         fontType;       /* font type */
   short         firstChar;      /* ASCII code of first character */
   short         lastChar;       /* ASCII code of last character */
   short         widMax;         /* maximum character width */
   short         kernMax;        /* negative of maximum character kern */
   short         nDescent;       /* negative of descent */
   short         fRectWidth;     /* width of font rectangle */
   short         fRectHeight;    /* height of font rectangle */
   short         owTLoc;         /* offset to offset/width table */
   short         ascent;         /* ascent */
   short         descent;        /* descent */
   short         leading;        /* leading */
   short         rowWords;       /* row width of bit image / 2 */
   /*
   short         bitImage[(rowWords-1)+1][(fRectHeight-1)+1];
   short         locTable[(lastChar+2-firstChar)+1];
   short         owTable[(lastChar+2-firstChar)+1];
   */
} FontRec;


pascal void InitFonts()
void GetFontName(fontNum,theName)
   short fontNum;
   char *theName;
void GetFNum(fontName,theNum)
   char *fontName;
   short *theNum;
pascal Boolean RealFont(fontNum,size)
   short fontNum;
   short size;
pascal void SetFontLock(lockFlag)
   Boolean lockFlag;
pascal FMOutPtr SwapFont(inRec)
   FMInput *inRec;
```

DESCRIPTION

The Font Manager supports the character fonts used to draw text with QuickDraw. For more detailed information see the Font Manager chapter of *Inside Macintosh.*

NAME
memory—Memory Manager

SYNOPSIS
```
#include <types.h>
#include <memory.h>

#define maxSize   0x800000   /* maximum block size */

/* Result Codes */

#define memFullErr   (-108)  /* not enough room in zone */
#define memLockedErr (-117)  /* block is locked */
#define memPurErr    (-112)  /* attempt to purge a locked block */
#define memWZErr     (-111)  /* attempt to operate on a free block */
#define nilHandleErr (-109)  /* nil master pointer */
#define noErr           0    /* no error */

typedef long Size;

typedef struct Zone {
   Ptr      bkLim;          /* limit pointer */
   Ptr      purgePtr;       /* used internally */
   Ptr      hFstFree;       /* first free master pointer */
   long     zcbFree;        /* number of free bytes */
   ProcPtr  gzProc;         /* grow zone function */
   short    moreMast;       /* master pointers to allocate */
   short    flags;          /* used internally */
   short    cntRel;         /* relocatable blocks */
   short    maxRel;         /* maximum cntRel value */
   short    cntNRel;        /* nonrelocatable blocks */
   short    maxNRel;        /* maximum maxRel value */
   short    cntEmpty;       /* empty master pointers */
   short    cntHandles;     /* total master pointers */
   long     minCBFree;      /* minimum zcbFree value */
   ProcPtr  purgeProc;      /* purge warning procedure */
   Ptr      sparePtr;       /* used internally */
   Ptr      allocPtr;       /* used internally */
   short    heapData;       /* first usable byte in zone */
} Zone, *THz;


/* Initialization and Allocation */

void InitApplZone()
void SetApplBase(startPtr)
   Ptr startPtr;
void InitZone(pGrowProc,cMoreMasters,limitPtr,startPtr)
   ProcPtr pGrowProc;
   short cMoreMasters;
   Ptr limitPtr,startPtr;
void SetApplLimit(zoneLimit)
   Ptr zoneLimit;
Ptr GetApplLimit()
void MaxApplZone()
void MoreMasters()
```

```
/* Heap Zone Access */

THz GetZone()
void SetZone(hz)
   THz hz;
THz SystemZone()
THz ApplicZone()


/* Allocating and Releasing Relocatable Blocks */

Handle NewHandle(logicalSize)
   Size logicalSize;
void DisposHandle(h)
   Handle h;
Size GetHandleSize(h)
   Handle h;
void SetHandleSize(h,newSize)
   Handle h;
   Size newSize;
THz HandleZone(h)
   Handle h;
Handle RecoverHandle(p)
   Ptr p;
void ReallocHandle(h,logicalSize)
   Handle h;
   Size logicalSize;


/* Allocating and Releasing Nonrelocatable Blocks */

Ptr NewPtr(logicalSize)
   Size logicalSize;
void DisposPtr(p)
   Ptr p;
Size GetPtrSize(p)
   Ptr p;
void SetPtrSize(p,newSize)
   Ptr p;
   Size newSize;
THz PtrZone(p)
   Ptr p;


/* Freeing Space in the Heap */

long FreeMem()
Size MaxMem(grow)
   Size *grow;
Size CompactMem(cbNeeded)
   Size cbNeeded;
void ResrvMem(cbNeeded)
   Size cbNeeded;
void PurgeMem(cbNeeded)
   Size cbNeeded;
void EmptyHandle(h)
```

```
   Handle h;


/* Properties of Relocatable Blocks */

void HLock(h)
   Handle h;
void HUnlock(h)
   Handle h;
void HPurge(h)
   Handle h;
void HNoPurge(h)
   Handle h;


/* Grow Zone Operations */

void SetGrowZone(growZone)
   ProcPtr growZone;
Boolean GZCritical()
Handle GZSaveHnd()


/* Miscellaneous Routines */

BlockMove(sourcePtr,destPtr,byteCount)
   Ptr sourcePtr, destPtr;
   long byteCount;
Ptr TopMem()
void MoveHHi(h)
   Handle h;
OSErr MemError()
```

USER ROUTINES

```
pascal Size MyGrowZone(cbNeeded)
   Size cbNeeded;
```

DESCRIPTION

The Memory Manager provides dynamic allocation of both relocatable and nonrelocatable memory space within the system and application heaps.

For more detailed information see the Memory Manager chapter of *Inside Macintosh.*

NAME
menus—Menu Manager

SYNOPSIS
```
#include <types.h>
#include <menus.h>

/* Special Characters */

#define noMark           '\0'

/* Messages to Menu Definition Functions */

#define mDrawMsg          0
#define mChooseMsg        1
#define mSizeMsg          2

/* Resource ID of Standard Menu Definition Procedure */

#define textMenuProc      0


typedef struct MenuInfo {
   short      menuID;
   short      menuWidth;
   short      menuHeight;
   ProcHandle menuProc;
   long       enableFlags;
   Str255     menuData;
} MenuInfo, *MenuPtr, **MenuHandle;


/* Initialization and Allocation */

pascal void InitMenus()
MenuHandle NewMenu(menuID,menuTitle)
   short menuID;
   char *menuTitle;
pascal MenuHandle GetMenu(resourceID)
   short resourceID;
pascal void DisposeMenu(theMenu)
   MenuHandle theMenu;
void AppendMenu(theMenu,data)
   MenuHandle theMenu;
   char *data;
pascal void AddResMenu(theMenu,theType)
   MenuHandle theMenu;
   ResType theType;
pascal void InsertResMenu(theMenu,theType,afterItem)
   MenuHandle theMenu;
   ResType theType;
   short afterItem;


/* Forming the Menu Bar */

pascal void InsertMenu(theMenu,beforeID)
```

```
   MenuHandle theMenu;
   short beforeID;
pascal void DrawMenuBar()
pascal void DeleteMenu(menuID)
   short menuID;
pascal void ClearMenuBar()
pascal Handle GetNewMBar(menuBarID)
   short menuBarID;
pascal Handle GetMenuBar()
pascal void SetMenuBar(menuList)
   Handle menuList;


/* Choosing From a Menu */

long MenuSelect(startPt)
   Point *startPt;
pascal long MenuKey(ch)
   short ch;
pascal void HiliteMenu(menuID)
   short menuID;


/* Controlling Items' Appearance */

void SetItem(theMenu,item,itemString)
   MenuHandle theMenu;
   short item;
   char *itemString;
void GetItem(theMenu,item,itemString)
   MenuHandle theMenu;
   short item;
   char *itemString;
pascal void DisableItem(theMenu,item)
   MenuHandle theMenu;
   short item;
pascal void EnableItem(theMenu,item)
   MenuHandle theMenu;
   short item;
pascal void CheckItem(theMenu,item,checked)
   MenuHandle theMenu;
   short item;
   Boolean checked;
pascal void SetItemMark(theMenu,item,markChar)
   MenuHandle theMenu;
   short item;
   short markChar;
pascal void GetItemMark(menu,item,markChar)
   MenuHandle menu;
   short item;
   short *markChar;
pascal void SetItemIcon(theMenu,item,iconNum)
   MenuHandle theMenu;
   short item;
   short iconNum;
pascal void GetItemIcon(theMenu,item,iconNum)
   MenuHandle theMenu;
```

```
      short item;
      short *iconNum;
   pascal void SetItemStyle(theMenu,item,chStyle)
      MenuHandle theMenu;
      short item;
      Style chStyle;
   pascal void GetItemStyle(menu,item,chStyle)
      MenuHandle menu;
      short item;
      Style *chStyle;

   /* Miscellaneous Utilities */

   pascal void CalcMenuSize(theMenu)
      MenuHandle theMenu;
   pascal short CountMItems(theMenu)
      MenuHandle theMenu;
   pascal MenuHandle GetMHandle(menuID)
      short menuID;
   pascal void FlashMenuBar(menuID)
      short menuID;
   pascal void SetMenuFlash(count)
      short count;
```

USER ROUTINES

```
   pascal void MyMenu(message,theMenu,menuRect,hitPt,whichItem)
      short message;
      MenuHandle theMenu;
      Rect *menuRect;
      Point hitPt;
      short *whichItem;
```

DESCRIPTION

The Menu Manager provides routines for creating and using menus.

Warning: The names of desk accessories start with a null byte. The output parameter from GetMenuItem will return a string that begins with a null byte when a desk accessory is selected from the Apple menu. OpenDeskAcc skips over the null byte when interpreting its parameter.

For more detailed information see the Menu Manager chapter of *Inside Macintosh.*

NAME

osevents—Operating System Event Manager

SYNOPSIS

```
#include <types.h>
#include <osevents.h>

typedef struct EvQEl {
   struct QElem *qLink;
   short    qType;
   short    evtQWhat;
   long     evtQMessage;
   long     evtQWhen;
   Point    evtQWhere;
   short    evtQModifiers;
} EvQEl;


/* Setting the System Event Mask */

void SetEventMask(theMask)
   short theMask;


/* Posting and Removing Events */

OSErr PostEvent(eventCode,eventMsg)
   short eventCode;
   long eventMsg;
void FlushEvents(eventMask,stopMask)
   short eventMask;
   short stopMask;


/* Accessing Events */

Boolean GetOSEvent(eventMask,theEvent)
   short eventMask;
   struct EventRecord *theEvent;
Boolean OSEventAvail(eventMask,theEvent)
   short eventMask;
   struct EventRecord *theEvent;


/* Directly Accessing the Event Queue */

struct QHdr *GetEvQHdr()
```

DESCRIPTION

The Operating System Event Manager provides a low-level interface to the Macintosh keyboard, keypad, and mouse.

For more detailed information see the Operating System Event Manager chapter of *Inside Macintosh.*

NAME

osutils—Operating System Utilities

SYNOPSIS

```
#include <types.h>
#include <osutils.h>


#define  noErr            0    /* no error */
#define  qErr            (-1)  /* element not in specified queue */
#define  clkRdErr       (-85)  /* unable to read clock */
#define  clkWrErr       (-86)  /* time written did not verify */
#define  prWrErr        (-87)  /* parameter RAM written did not verify */
#define  prInitErr      (-88)  /* validity status is not 0xa8 */
#define  memFullErr    (-108)  /* not enough room in heap */
#define  nilHandleErr  (-109)  /* nil master pointer */
#define  memWZErr      (-111)  /* attempt to operate on a free block */

/* Serial Port configuration constants for config field of SysParmType */

#define  useFree          0    /* use undefined */
#define  useATalk         1    /* AppleTalk */
#define  useAsync         2    /* Async */

/* Machine type returned by "Environs" routine */

#define  macXLMachine     0    /* Macintosh XL (a.k.a. Lisa) */
#define  macMachine       1    /* Macintosh */

/* typedef long  OSType;   appears in file TYPES */
/* typedef short OSErr;    appears in file TYPES */


typedef struct SysParmType {
   char       valid;           /* validity status */
   char       aTalkA;          /* AppleTalk node # hint for port A */
   char       aTalkB;          /* AppleTalk node # hint for port B */
   char       config;          /* AppleTalk serial port configuration */
                               /*    port A = bits 4-7, B = bits 0-3 */
   short      portA;           /* modem port configuration */
   short      portB;           /* printer port configuration */
   long       alarm;           /* alarm setting */
   short      font;            /* default application font number - 1 */
   short      kbdPrint;        /* auto-key threshold and rate, printer */
                               /*    connection */
   short      volClik;         /* speaker volume, double-click and */
                               /*    caret-blink times */
   short      misc;            /* mouse scaling, system startup disk, */
                               /*    menu blink */
} SysParmType, *SysPPtr;


typedef struct QElem {
   struct QElem    *qLink;     /* next queue entry */
   short           qType;      /* queue type */
   short           qData[0];   /* queue type specific data */
} QElem, *QElemPtr;
```

```
typedef struct QHdr {
   short      qFlags;            /* queue flags */
   QElemPtr   qHead;             /* first queue entry */
   QElemPtr   qTail;             /* last queue entry */
} QHdr, *QHdrPtr;


typedef enum {
   dummyType,
   vType,                        /* vertical retrace queue type */
   ioQType,                      /* file I/O or driver I/O type */
   drvQType,                     /* drive queue type */
   evType,                       /* event queue type */
   fsQType                       /* volume-control-block queue type */
} QTypes;


typedef struct DateTimeRec {
   short      year;              /* four-digit year */
   short      month;             /* 1 to 12 for January to December */
   short      day;               /* 1 to 31 */
   short      hour;              /* 0 to 23 */
   short      minute;            /* 0 to 59 */
   short      second;            /* 0 to 59 */
   short      dayOfWeek;         /* 1 to 7 for Sunday to Saturday */
} DateTimeRec;


/* Pointer and Handle Manipulation */

OSErr HandToHand(theHndl)
   Handle *theHndl;
OSErr PtrToHand(srcPtr,dstHndl,size)
   Ptr srcPtr;
   Handle *dstHndl;
   long size;
OSErr PtrToXHand(srcPtr,dstHndl,size)
   Ptr srcPtr;
   Handle dstHndl;
   long size;
OSErr HandAndHand(aHndl,bHndl)
   Handle aHndl,bHndl;
OSErr PtrAndHand(pntr,hndl,size)
   Ptr pntr;
   Handle hndl;
   long size;


/* String Comparison */

Boolean EqualString(aStr,bStr,caseSens,diacSens)
   char *aStr,*bStr;
   Boolean caseSens,diacSens;
void UprString(theString,diacSens)
   char *theString;
   Boolean marks;


/* Date and Time Operations */

OSErr ReadDateTime(secs)
   long *secs;
void GetDateTime(secs)
   long *secs;
OSErr SetDateTime(secs)
   long secs;
void Date2Secs(date,secs)
   DateTimeRec *date;
   long *secs;
void Secs2Date(secs,date)
   long secs;
   DateTimeRec *date;
void GetTime(date)
   DateTimeRec *date;
void SetTime(date)
   DateTimeRec *date;


/* Parameter RAM Operations */

OSErr InitUtil()
SysPPtr GetSysPPtr()
OSErr WriteParam()


/* Queue Manipulation */

void Enqueue(qElement,theQueue)
   QElemPtr qElement;
   QHdrPtr theQueue;
OSErr Dequeue(qElement,theQueue)
   QElemPtr qElement;
   QHdrPtr theQueue;


/* Dispatch Table Utilities */

void SetTrapAddress(trapAddr,trapNum)
   long trapAddr;
   short trapNum;
long GetTrapAddress(trapNum)
   short trapNum;


/* Miscellaneous Utilities */

void Delay(numTicks,finalTicks)
   long numTicks,*finalTicks;
pascal void SysBeep(duration)
   short duration;
void Environs(rom,machine)
   short *rom,*machine;
```

```
void Restart()
```

DESCRIPTION

The Operating System Utilities are a set of routines and data types in the Operating System that perform generally useful operations such as manipulating pointers and handles, comparing strings, and reading the date and time.

For more detailed information see the Operating System Utilities chapter of *Inside Macintosh.*

NAME

packages—
Package Manager, Disk Initialization, Standard File Package, International Utilities, Binary-Decimal Conversion

SYNOPSIS

```
#include <types.h>
#include <packages.h>


/* Package Manager ------------------------------------------- */

/* Resource IDs for packages */

#define dskInit         2       /* Disk Initializaton */
#define stdFile         3       /* Standard File */
#define flPoint         4       /* Floating-Point Arithmetic */
#define trFunc          5       /* Transcendental Functions */
#define intUtil         6       /* International Utilities */
#define bdConv          7       /* Binary/Decimal Conversion */

pascal void InitAllPacks()
pascal void InitPack(packID)
   short packID;


/* Disk Initialization Package ------------------------------- */

void DILoad()
void DIUnLoad()
short DIBadMount(where,evtMessage)
   Point where;
   long evtMessage;
OSErr DIFormat(drvNum)
   short drvNum;
OSErr DIVerify(drvNum)
   short drvNum;
OSErr DIZero(drvNum,volName)
   short drvNum;
   char *volName;


/* Standard File Package ------------------------------------- */

#define putDlgID   (-3999)    /* SFPutFile dialog template ID */

#define putSave         1     /* Save button */
#define putCancel       2     /* Cancel button */
#define putEject        5     /* Eject button */
#define putDrive        6     /* Drive button */
#define putName         7     /* editTExt item for file name */

#define getDlgID   (-4000)    /* SFGetFile dialog template ID */

#define getOpen         1     /* Open button */
#define getCancel       3     /* Cancel button */
#define getEject        5     /* Eject button */
```

```
#define getDrive        6     /* Drive button */
#define getNmList       7     /* userItem for file name list */
#define getScroll       8     /* userItem for scroll bar */

typedef struct SFReply {
   Boolean good;              /* FALSE if ignore command */
   Boolean copy;              /* not used */
   OSType fType;              /* file type or not used */
   short vRefNum;             /* volume reference number */
   short version;             /* file's version number */
   String(63) fName;          /* file name */
} SFReply;

typedef OSType SFTypeList[4];

void SFPutFile(where,prompt,origName,dlgHook,reply)
   Point *where;
   char *prompt;
   char *origName;
   ProcPtr dlgHook;
   SFReply *reply;
void SFPPutFile(where,prompt,origName,dlgHook,reply,dlgID,filterProc)
   Point *where;
   char *prompt;
   char *origName;
   ProcPtr dlgHook;
   SFReply *reply;
   short dlgID;
   ProcPtr filterProc;
void SFGetFile(where,prompt,fileFilter,numTypes,typeList,dlgHook,reply)
   Point *where;
   char *prompt;
   ProcPtr fileFilter;
   short numTypes;
   SFTypeList typeList;
   ProcPtr dlgHook;
   SFReply *reply;
void SFPGetFile(where,prompt,fileFilter,numTypes,typeList,dlgHook,reply,
      dlgID,filterProc)
   Point *where;
   char *prompt;
   ProcPtr fileFilter;
   short numTypes;
   SFTypeList typeList;
   ProcPtr dlgHook;
   SFReply *reply;
   short dlgID;
   ProcPtr filterProc;


/* International Utilities Package ---------------------------- */

/* Masks for currency format */

#define currSymLead    16     /* set if currency symbol leads */
#define currNegSym     32     /* set if minus sign for negative */
#define currTrailingZ  64     /* set if trailing decimal zeroes */
```

```
#define currLeadingZ  128     /* set if leading integer zeroes */

/* Order of short date elements */

#define mdy             0     /* month day year */
#define dmy             1     /* day month year */
#define ymd             2     /* year month day */

/* Masks for short date format */

#define dayLdingZ      32     /* set if leading zero for day */
#define mntLdingZ      64     /* set if leading zero for month */
#define century       128     /* set if century included */

/* Masks for time format */

#define secLeadingZ    32     /* set if leading zero for seconds */
#define minLeadingZ    64     /* set if leading zero for minutes */
#define hrLeadingZ    128     /* set if leading zero for hours */

/* Country Codes -- high-order byte of version information */

#define verUS           0
#define verFrance       1
#define verBritain      2
#define verGermany      3
#define verItaly        4
#define verNetherlands  5
#define verBelgiumLux   6
#define verSweden       7
#define verSpain        8
#define verDenmark      9
#define verPortugal    10
#define verFrCanada    11
#define verNorway      12
#define verIsrael      13
#define verJapan       14
#define verAustralia   15
#define verArabia      16
#define verFinland     17
#define verFrSwiss     18
#define verGrSwiss     19
#define verGreece      20
#define verIceland     21
#define verMalta       22
#define verCyprus      23
#define verTurkey      24
#define verYugoslavia  25

typedef struct Intl0Rec {
   char decimalPt;            /* decimal point character */
   char thousSep;             /* thousands separator */
   char listSep;              /* list separator */
   char currSym1;             /* currency symbols (3 bytes long) */
   char currSym2;
   char currSym3;
   unsigned char currFmt;     /* currency format */
   unsigned char dateOrder;   /* short date order - dmy, ymd or mdy */
   unsigned char shrtDateFmt; /* short date format */
   char dateSep;              /* date separator */
   unsigned char timeCycle;   /* 0 if 24-hour cycle, 255 if 1- hour */
   unsigned char timeFmt;     /* time format */
   char mornStr[4];           /* trailing string for first 12 hours */
   char eveStr[4];            /* trailing stringfor last 12 hours */
   char timeSep;              /* time separator */
   char time1Suff;            /* trailing string for 24-hour cycle */
   char time2Suff;
   char time3Suff;
   char time4Suff;
   char time5Suff;
   char time6Suff;
   char time7Suff;
   char time8Suff;
   unsigned char metricSys;   /* 255 for metric, 0 if not */
   short intl0Vers;           /* version information - country, vers */
} Intl0Rec, *Intl0Ptr, **Intl0Hndl;

typedef struct Intl1Rec {
   String(15) days[7];        /* day names */
   String(15) months[12];     /* month names */
   unsigned char suppressDay; /* 0 for day name, 255 for none */
   unsigned char lngDateFmt;  /* order of long date elements */
   unsigned char dayLeading0; /* 255 for leading 0 in day number */
   unsigned char abbrLen;     /* length for abbreaviating names */
   char st0[4];               /* date punctuation  */
   char st1[4];
   char st2[4];
   char st3[4];
   char st4[4];
   short intl1Vers;           /* version information */
   short localRtn[0];         /* routine for string conparison */
} Intl1Rec, *Intl1Ptr, **Intl1Hndl;

typedef enum {shortDate,longDate,abbrevDate} DateForm;

void IUDateString(dateTime,form,result)
   long dateTime;
   DateForm form;
   char *result;
void IUDatePString(dateTime,form,result,intlParam)
   long dateTime;
   DateForm form;
   char *result;
   Handle intlParam;
void IUTimeString(dateTime,wantSeconds,result)
   long dateTime;
   Boolean wantSeconds;
   char *result;
void IUTimePString(dateTime,wantSeconds,result,intlParam)
   long dateTime;
   Boolean wantSeconds;
   char *result;
   Handle intlParam;
Boolean IUMetric()
```

```
Handle IUGetIntl(theID)
   short theID;
void IUSetIntl(refNum,theID,intlParam)
   short refNum;
   short theID;
   Handle intlParam;
short IUCompString(aStr,bStr)
   char *aStr;
   char *bStr;
short IUMagString(aPtr,bPtr,aLen,bLen)
   Ptr aPtr,bPtr;
   short aLen,bLen;
short IUEqualString(aStr,bStr)
   char *aStr;
   char *bStr;
short IUMagIDString(aPtr,bPtr,aLen,bLen)
   Ptr aPtr,bPtr;
   short aLen,bLen;


/* Binary-Decimal Conversion Package -------------------------- */

void StringToNum(theString,theNum)
   char *theString;
   long *theNum;
void NumToString(theNum,theString)
   long theNum;
   char *theString;
```

USER ROUTINES

```
/* Standard File Package -------------------------------------- */

short MyDlg(item,theDialog)
   short item;
   DialogPtr theDialog;

Boolean MyFileFilter(paramBlock)
   ParmBlkPtr paramBlock;
```

DESCRIPTION

The Package Manager provides for the initialization of packages.

For more detailed information see the following chapters of *Inside Macintosh*: Package Manager, Disk Initialization Package, Standard File Package, International Utilities Package, Binary-Decimal Conversion Package.

NAME

printing—Printing Manager

SYNOPSIS

```
#include <types.h>
#include <quickdraw.h>
#include <printing.h>



/* Printing Methods */

#define  bDraftLoop           0  /* draft printing */
#define  bSpoolLoop           1  /* spooling */

/* Printer specification in prStl field of print record */

#define  bDevCItoh            1  /* ImageWriter printer */
#define  bDevLaser            3  /* LaserWriter printer */

/* Maximum number of pages in a spool file */

#define  iPFMaxPgs          128  /* max pages in a spool file */
#define  iPrPgFract         120  /* paper units per inch */

/* Result codes */

#define  noErr                0  /* no error */
#define  iPrSavPFil        (-1)  /* saving spool file */
#define  iIOAbort         (-27)  /* I/O abort error */
#define  iMemFullErr     (-108)  /* not enough room in heap zone */
#define  iPrAbort           128  /* application or user requested abort */

/* Printer Driver Control Call Parameters */

#define  iPrDevCtl            7  /* device control */
#define  lPrReset    0x00010000  /* reset printer */
#define  lPrLineFeed 0x00030000  /* start new line */
#define  lPrLFSixth  0x0003FFFF  /* standard 1/6" line feed */
#define  lPrPageEnd  0x00020000  /* start new page */
#define  iPrBitsCtl           4  /* bit map printing */
#define  lScreenBits          0  /* configurable */
#define  lPaintBits           1  /* 72 x 72 dots */
#define  iPrIOCtl             5  /* text streaming */

/* Printing Resources */

#define  sPrDrvr       ".Print" /* Printer Driver resource name */
#define  iPrDrvrRef        (-3) /* Printer Driver reference number */


/* Type definitions */

typedef Rect *TPRect;

typedef struct TPrPort {
   GrafPort gPort;      /* graph port to draw in */
   QDProcs  gProcs;      /* pointers to drawing routines */
   long     lGParam1;    /* internal */
   long     lGParam2;    /* internal */
   long     lGParam3;    /* internal */
   long     lGParam4;    /* internal */
   Boolean  fOurPtr;     /* internal */
   Boolean  fOurBits;    /* internal */
} TPrPort, *TPPrPort;

typedef struct TPrInfo {
   short    iDev;        /* printer information */
   short    iVRes;       /* printer vertical resolution */
   short    iHRes;       /* printer horizontal resolution */
   Rect     rPage;       /* page rectangle */
} TPrInfo;

typedef enum {feedCut,feedFanfold,feedMechCut,feedOther} TFeed;

typedef struct TPrStl {
   short    wDev;        /* high byte specifies device */
   short    iPageV;      /* paper height */
   short    iPageH;      /* paper width */
   char     bPort;       /* printer or modem port - ignored */
   TFeed    feed;        /* paper type */
} TPrStl;

typedef enum {scanTB,scanBT,scanLR,scanRL} TScan;

typedef struct TPrXInfo {
   short    iRowBytes;   /* bytes per row */
   short    iBandV;      /* vertical dots */
   short    iBandH;      /* horizontal dots */
   short    iDevBytes;   /* size of bit image */
   short    iBands;      /* bands per page */
   char     bPatScale;   /* used by QuickDraw */
   char     bUlThick;    /* underline thickness */
   char     bUlOffset;   /* underline offset */
   char     bUlShadow;   /* underline descender */
   TScan    scan;        /* scan direction */
   char     bXInfoX;     /* not used */
} TPrXInfo;

typedef struct TPrJob {
   short    iFstPage;    /* first page to print */
   short    iLstPage;    /* last page to print */
   short    iCopies;     /* number of copies */
   char     bJDocLoop;   /* printing method */
   Boolean  fFromUsr;    /* true if called from application */
   ProcPtr  pIdleProc;   /* background procedure */
   StringPtr pFileName;  /* spool file name */  ???
   short    iFileVol;    /* volume reference number */
   char     bFileVers;   /* version number of spool file */
   char     bJobX;       /* not used */
} TPrJob;

typedef struct TPrint {
   short    iPrVersion; /* Printing Manager version */
   TPrInfo  prInfo;      /* printing information */
   Rect     rPaper;      /* paper rectangle */
   TPrStl   prStl;       /* style information */
   TPrInfo  prInfoPT;    /* copy of prInfo */
   TPrXInfo prXInfo;     /* band information */
   TPrJob   prJob;       /* job information */
   short    printX[19]   /* internal */
} TPrint, *TPPrint, **THPrint;

typedef struct TPrStatus {
   short    iTotPages;   /* total number of pages */
   short    iCurPage;    /* page being printed */
   short    iTotCopies;  /* number of copies */
   short    iCurCopy;    /* copy begin printed */
   short    iTotBands;   /* bands per page */
   short    iCurBand;    /* band being printed */
   Boolean  fPgDirty;    /* true if started printing page */
   Boolean  fImaging;    /* true if imaging */
   THPrint  hPrint;      /* print record */
   TPPrPort pPrPort;     /* printing port */
   PicHandle hPic;       /* internal */
} TPrStatus;


/* Initialization and Termination */

pascal void PrOpen()
pascal void PrClose()

/* Print Records and Dialogs */

pascal void PrintDefault(hPrint)
   THPrint hPrint;
pascal Boolean PrValidate(hPrint)
   THPrint hPrint;
pascal Boolean PrStlDialog(hPrint)
   THPrint hPrint;
pascal Boolean PrJobDialog(hPrint)
   THPrint hPrint;
pascal void PrJobMerge(hPrintSrc,hPrintDst)
   THPrint hPrintSrc,hPrintDst;

/* Document Printing */

pascal TPPrPort PrOpenDoc(hPrint,pPrPort,pIOBuf)
   THPrint hPrint;
   TPPrPort pPrPort;
   Ptr pIOBuf;
pascal void PrCloseDoc(pPrPort)
   TPPrPort pPrPort;
pascal void PrOpenPage(pPrPort,pPageFrame)
   TPPrPort pPrPort;
   TPRect pPageFrame;
pascal void PrClosePage(pPrPort)
   TPPrPort pPrPort;

/* Spool Printing */
```

```
pascal void PrPicFile(hPrint,pPrPort,pIOBuf,pDevBuf,prStatus)
   THPrint hPrint;
   TPPrPort pPrPort;
   Ptr pIOBuf,pDevBuf;
   TPrStatus *prStatus;

/* Handling Errors */

pascal short PrError()
pascal void PrSetError(iErr)
   short iErr;

/* Low Level Driver Access */

pascal void PrDrvrOpen()
pascal void PrDrvrClose()
pascal void PrCtlCall(iWhichCtl,lParam1,lParam2,lParam3)
   short iWhichCtl;
   long lParam1,lParam2,lParam3;
pascal Handle PrDrvrDCE()
pascal short PrDrvrVers()
```

DESCRIPTION

The Printing Manager supports printing on a variety of devices.

For more detailed information see the Printing Manager chapter of *Inside Macintosh.*

NOTE

The current Pascal implementation has additional constants and data types that aren't documented in *Inside Macintosh* because they're not generally used. This interface follows *Inside Macintosh* .

Programs that call the high-level printing routines must be linked with object file prlink.obj. Programs that call the low-level driver routines must link with prscreen.obj. Linking with both files yields duplicate definitions. Therefore a program can't call both the high-level routines and the driver routines.

NAME
quickdraw—QuickDraw

SYNOPSIS
```
#include <types.h>
#include <quickdraw.h>

/* 16 Transfer Modes */

#define  srcCopy       0
#define  srcOr         1
#define  srcXor        2
#define  srcBic        3
#define  notSrcCopy    4
#define  notSrcOr      5
#define  notSrcXor     6
#define  notSrcBic     7
#define  patCopy       8
#define  patOr         9
#define  patXor        10
#define  patBic        11
#define  notPatCopy    12
#define  notPatOr      13
#define  notPatXor     14
#define  notPatBic     15

/* QuickDraw Color Separation Constants */

#define  normalBit     0
#define  inverseBit    1
#define  redBit        4       /* RGB Additive Mapping */
#define  greenBit      3
#define  blueBit       2
#define  cyanBit       8       /* CMYBk Subtractive Mapping */
#define  magentaBit    7
#define  yellowBit     6
#define  blackBit      5
#define  blackColor    33      /* Colors Expressed in these Mappings */
#define  whiteColor    30
#define  redColor      205
#define  greenColor    341
#define  blueColor     409
#define  cyanColor     273
#define  magentaColor  137
#define  yellowColor   69

/* Picture Comments */

#define  picLParen     0
#define  picRParen     1

/* Type Style Constants */

#define  normal        0x00
#define  bold          0x01
#define  italic        0x02
#define  underline     0x04
```

```
#define  outline       0x08
#define  shadow        0x10
#define  condense      0x20
#define  expand        0x40


/* Types */

typedef  unsigned char Pattern[8];
typedef  short Bits16[16];
typedef  enum {frame,paint,erase,invert,fill} GrafVerb;

/* typedefs Style, Point, and Rect appear in file TYPES */

typedef struct FontInfo {
   short    ascent;
   short    descent;
   short    widMax;
   short    leading;
} FontInfo;

typedef struct BitMap {
   Ptr      baseAddr;
   short    rowBytes;
   Rect     bounds;
} BitMap;

typedef struct Cursor {
   Bits16   data;
   Bits16   mask;
   Point    hotSpot;
} Cursor;

typedef struct PenState {
   Point    pnLoc;
   Point    pnSize;
   short    pnMode;
   Pattern  pnPat;
} PenState;

typedef struct Region {
   short    rgnSize;
   Rect     rgnBBox;
   short    rgnData[0];
} Region, *RgnPtr, **RgnHandle;

typedef struct Picture {
   short    picSize;
   Rect     picFrame;
   short    picData[0];
} Picture, *PicPtr, **PicHandle;

typedef struct Polygon {
   short    polySize;
   Rect     polyBBox;
   Point    polyPoints[0];
} Polygon, *PolyPtr, **PolyHandle;
```

```
typedef struct QDProcs {
   ProcPtr  textProc;
   ProcPtr  lineProc;
   ProcPtr  rectProc;
   ProcPtr  rRectProc;
   ProcPtr  ovalProc;
   ProcPtr  arcProc;
   ProcPtr  polyProc;
   ProcPtr  rgnProc;
   ProcPtr  bitsProc;
   ProcPtr  commentProc;
   ProcPtr  txMeasProc;
   ProcPtr  getPicProc;
   ProcPtr  putPicProc;
} QDProcs, *QDProcsPtr;

typedef struct GrafPort {
   short       device;
   BitMap      portBits;
   Rect        portRect;
   RgnHandle   visRgn;
   RgnHandle   clipRgn;
   Pattern     bkPat;
   Pattern     fillPat;
   Point       pnLoc;
   Point       pnSize;
   short       pnMode;
   Pattern     pnPat;
   short       pnVis;
   short       txFont;
   Style       txFace;
   short       txMode;
   short       txSize;
   long        spExtra;
   long        fgColor;
   long        bkColor;
   short       colrBit;
   short       patStretch;
   PicHandle   picSave;
   RgnHandle   rgnSave;
   PolyHandle  polySave;
   QDProcsPtr  grafProcs;
} GrafPort, *GrafPtr;


/* External Variable Declarations */

extern struct qd {
   char      private[78];
   long      randSeed;
   BitMap    screenBits;
   Cursor    arrow;
   Pattern   dkGray;
   Pattern   ltGray;
   Pattern   gray;
   Pattern   black;
   Pattern   white;
   GrafPtr   thePort;
} qd;


/* GrafPort Routines */

pascal void InitGraf(globalPtr)
   Ptr globalPtr;
pascal void OpenPort(port)
   GrafPtr port;
pascal void InitPort(port)
   GrafPtr port;
pascal void ClosePort(port)
   GrafPtr port;
pascal void SetPort(port)
   GrafPtr port;
pascal void GetPort(port)
   GrafPtr *port;
pascal void GrafDevice(device)
   short device;
pascal void SetPortBits(bm)
   BitMap *bm;
pascal void PortSize(width,height)
   short width,height;
pascal void MovePortTo(leftGlobal,rightGlobal)
   short leftGlobal,rightGlobal;
pascal void SetOrigin(h,v)
   short h,v;
pascal void SetClip(rgn)
   RgnHandle rgn;
pascal void GetClip(rgn)
   RgnHandle rgn;
pascal void ClipRect(r)
   Rect *r;
pascal void BackPat(pat)
   Pattern *pat;


/* Cursor Routines */

pascal void InitCursor()
pascal void SetCursor(crsr)
   Cursor *crsr;
pascal void HideCursor()
pascal void ShowCursor()
pascal void ObscureCursor()


/* Line Routines */

pascal void HidePen()
pascal void ShowPen()
pascal void GetPen(pt)
   Point *pt;
pascal void GetPenState(pnState)
   PenState *pnState;
```

```
pascal void SetPenState(pnState)
   PenState *pnState;
pascal void PenSize(width,height)
   short width,height;
pascal void PenMode(mode)
   short mode;
pascal void PenPat(pat)
   Pattern *pat;
pascal void PenNormal()
pascal void MoveTo(h,v)
   short h,v;
pascal void Move(dh,dv)
   short dh,dv;
pascal void LineTo(h,v)
   short h,v;
pascal void Line(dh,dv)
   short dh,dv;


/* Text Routines */

pascal void TextFont(font)
   short font;
pascal void TextFace(face)
   Style face;
pascal void TextMode(mode)
   short mode;
pascal void TextSize(size)
   short size;
pascal void SpaceExtra(extra)
   long extra;
pascal void DrawChar(ch)
   short ch;
void DrawString(s)
   char *s;
pascal void DrawText(textBuf,firstByte,byteCount)
   Ptr textBuf;
   short firstByte,byteCount;
pascal short CharWidth(ch)
   short ch;
short StringWidth(s)
   char *s;
pascal short TextWidth(textBuf,firstByte,byteCount)
   Ptr textBuf;
   short firstByte,byteCount;
pascal void GetFontInfo(info)
   FontInfo *info;


/* Drawing in Color */

pascal void ForeColor(color)
   long color;
pascal void BackColor(color)
   long color;
pascal void ColorBit(whichBit)
   short whichBit;
```

```
/* Rectangle Calculations */

pascal void SetRect(r,left,top,right,bottom)
   Rect *r;
   short left,top,right,bottom;
pascal void OffsetRect(r,dh,dv)
   Rect *r;
   short dh,dv;
pascal void InsetRect(r,dh,dv)
   Rect *r;
   short dh,dv;
pascal Boolean SectRect(srcRect1,srcRect2,dstRect)
   Rect *srcRect1,*srcRect2,*dstRect;
pascal void UnionRect(srcRect1,srcRect2,dstRect)
   Rect *srcRect1,*srcRect2,*dstRect;
Boolean PtInRect(pt,r)
   Point *pt;
   Rect *r;
void Pt2Rect(pt1,pt2,dstRect)
   Point *pt1, *pt2;
   Rect *dstRect;
void PtToAngle(r,pt,angle)
   Rect *r;
   Point *pt;
   short *angle;
pascal Boolean EqualRect(rect1,rect2)
   Rect *rect1,*rect2;
pascal Boolean EmptyRect(r)
   Rect *r;


/* Graphical Operations on Rectangles */

pascal void FrameRect(r)
   Rect *r;
pascal void PaintRect(r)
   Rect *r;
pascal void EraseRect(r)
   Rect *r;
pascal void InvertRect(r)
   Rect *r;
pascal void FillRect(r,pat)
   Rect *r;
   Pattern *pat;


/* Oval Routines */

pascal void FrameOval(r)
   Rect *r;
pascal void PaintOval(r)
   Rect *r;
pascal void EraseOval(r)
   Rect *r;
pascal void InvertOval(r)
```

```
   Rect *r;
pascal void FillOval(r,pat)
   Rect *r;
   Pattern *pat;


/* RoundRect Routines */

pascal void FrameRoundRect(r,ovalWidth,ovalHeight)
   Rect *r;
   short ovalWidth,ovalHeight;
pascal void PaintRoundRect(r,ovalWidth,ovalHeight)
   Rect *r;
   short ovalWidth,ovalHeight;
pascal void EraseRoundRect(r,ovalWidth,ovalHeight)
   Rect *r;
   short ovalWidth,ovalHeight;
pascal void InvertRoundRect(r,ovalWidth,ovalHeight)
   Rect *r;
   short ovalWidth,ovalHeight;
pascal void FillRoundRect(r,ovalWidth,ovalHeight,pat)
   Rect *r;
   short ovalWidth,ovalHeight;
   Pattern *pat;


/* Arc Routines */

pascal void FrameArc(r,startAngle,arcAngle)
   Rect *r;
   short startAngle,arcAngle;
pascal void PaintArc(r,startAngle,arcAngle)
   Rect *r;
   short startAngle,arcAngle;
pascal void EraseArc(r,startAngle,arcAngle)
   Rect *r;
   short startAngle,arcAngle;
pascal void InvertArc(r,startAngle,arcAngle)
   Rect *r;
   short startAngle,arcAngle;
pascal void FillArc(r,startAngle,arcAngle,pat)
   Rect *r;
   short startAngle,arcAngle;
   Pattern *pat;


/* Region Calculations */

pascal RgnHandle NewRgn()
pascal void DisposeRgn(rgn)
   RgnHandle rgn;
pascal void CopyRgn(srcRgn,dstRgn)
   RgnHandle srcRgn,dstRgn;
pascal void SetEmptyRgn(rgn)
   RgnHandle rgn;
pascal void SetRectRgn(rgn,left,top,right,bottom)
   RgnHandle rgn;
   short left,top,right,bottom;
pascal void RectRgn(rgn,r)
   RgnHandle rgn;
   Rect *r;
pascal void OpenRgn()
pascal void CloseRgn(dstRgn)
   RgnHandle dstRgn;
pascal void OffsetRgn(rgn,dh,dv)
   RgnHandle rgn;
   short dh,dv;
pascal void InsetRgn(rgn,dh,dv)
   RgnHandle rgn;
   short dh,dv;
pascal void SectRgn(srcRgnA,srcRgnB,dstRgn)
   RgnHandle srcRgnA,srcRgnB,dstRgn;
pascal void UnionRgn(srcRgnA,srcRgnB,dstRgn)
   RgnHandle srcRgnA,srcRgnB,dstRgn;
pascal void DiffRgn(srcRgnA,srcRgnB,dstRgn)
   RgnHandle srcRgnA,srcRgnB,dstRgn;
pascal void XorRgn(srcRgnA,srcRgnB,dstRgn)
   RgnHandle srcRgnA,srcRgnB,dstRgn;
Boolean PtInRgn(pt,rgn)
   Point *pt;
   RgnHandle rgn;
pascal Boolean RectInRgn(r,rgn)
   Rect *r;
   RgnHandle rgn;
pascal Boolean EqualRgn(rgnA,rgnB)
   RgnHandle rgnA,rgnB;
pascal Boolean EmptyRgn(rgn)
   RgnHandle rgn;


/* Graphical Operations on Regions */

pascal void FrameRgn(rgn)
   RgnHandle rgn;
pascal void PaintRgn(rgn)
   RgnHandle rgn;
pascal void EraseRgn(rgn)
   RgnHandle rgn;
pascal void InvertRgn(rgn)
   RgnHandle rgn;
pascal void FillRgn(rgn,pat)
   RgnHandle rgn;
   Pattern *pat;


/* Graphical Operations on Bit Maps */

pascal void ScrollRect(r,dh,dv,updateRgn)
   Rect *r;
   short dh,dv;
   RgnHandle updateRgn;
pascal void CopyBits(srcBits,dstBits,srcRect,dstRect,mode,maskRgn)
   BitMap *srcBits,*dstBits;
   Rect *srcRect,*dstRect;
   short mode;
   RgnHandle maskRgn;


/* Picture Routines */

pascal PicHandle OpenPicture(picFrame)
   Rect *picFrame;
pascal void PicComment(kind,dataSize,dataHandle)
   short kind,dataSize;
   Handle dataHandle;
pascal void ClosePicture()
pascal void DrawPicture(myPicture,dstRect)
   PicHandle myPicture;
   Rect *dstRect;
pascal void KillPicture(myPicture)
   PicHandle myPicture;


/* Polygon Calculations */

pascal PolyHandle OpenPoly()
pascal void ClosePoly()
pascal void KillPoly(poly)
   PolyHandle poly;
pascal void OffsetPoly(poly,dh,dv)
   PolyHandle poly;
   short dh,dv;


/* Graphical Operations on Polygons */

pascal void FramePoly(poly)
   PolyHandle poly;
pascal void PaintPoly(poly)
   PolyHandle poly;
pascal void ErasePoly(poly)
   PolyHandle poly;
pascal void InvertPoly(poly)
   PolyHandle poly;
pascal void FillPoly(poly,pat)
   PolyHandle poly;
   Pattern *pat;


/* Point Calculations */

void AddPt(srcPt,dstPt)
   Point *srcPt,*dstPt;
void SubPt(srcPt,dstPt)
   Point *srcPt,*dstPt;
pascal void SetPt(pt,h,v)
   Point *pt;
   short h,v;
Boolean EqualPt(pt1,pt2)
   Point *pt1, *pt2;
pascal void LocalToGlobal(pt)
```

```
   Point *pt;
pascal void GlobalToLocal(pt)
   Point *pt;


/* Miscellaneous Utility Routines */

pascal short Random()
pascal Boolean GetPixel(h,v)
   short h,v;
void StuffHex(thingPtr,s)
   Ptr thingPtr;
   char *s;
pascal void ScalePt(pt,srcRect,dstRect)
   Point *pt;
   Rect *srcRect,*dstRect;
pascal void MapPt(pt,srcRect,dstRect)
   Point *pt;
   Rect *srcRect,*dstRect;
pascal void MapRect(r,srcRect,dstRect)
   Rect *r,*srcRect,*dstRect;
pascal void MapRgn(rgn,srcRect,dstRect)
   RgnHandle rgn;
   Rect *srcRect,*dstRect;
pascal void MapPoly(poly,srcRect,dstRect)
   PolyHandle poly;
   Rect *srcRect,*dstRect;


/* Bottleneck Interface */

pascal void SetStdProcs(procs)
   QDProcsPtr procs;
void StdText(byteCount,textAddr,numer,denom)
   short byteCount;
   Ptr textAddr;
   Point *numer, *denom;
void StdLine(newPt)
   Point *newPt;
pascal void StdRect(verb,r)
   GrafVerb verb;
   Rect *r;
pascal void StdRRect(verb,r,ovalWidth,ovalHeight)
   GrafVerb verb;
   Rect *r;
   short ovalWidth,ovalHeight;
pascal void StdOval(verb,r)
   GrafVerb verb;
   Rect *r;
pascal void StdArc(verb,r,startAngle,arcAngle)
   GrafVerb verb;
   Rect *r;
   short startAngle,arcAngle;
pascal void StdPoly(verb,poly)
   GrafVerb verb;
   PolyHandle poly;
pascal void StdRgn(verb,rgn)
```

```
      GrafVerb verb;
      RgnHandle rgn;
   pascal void StdBits(srcBits,srcRect,dstRect,mode,maskRgn)
      BitMap *srcBits;
      Rect *srcRect,*dstRect;
      short mode;
      RgnHandle maskRgn;
   pascal void StdComment(kind,dataSize,dataHandle)
      short kind,dataSize;
      Handle dataHandle;
   pascal short StdTxMeas(byteCount,textAddr,numer,denom,info)
      short byteCount;
      Ptr textAddr;
      Point *numer,*denom;
      FontInfo *info;
   pascal void StdGetPic(dataPtr,byteCount)
      Ptr dataPtr;
      short byteCount;
   pascal void StdPutPic(dataPtr,byteCount)
      Ptr dataPtr;
      short byteCount;
```

USER ROUTINES

```
   pascal void MyText(byteCount,textAddr,numer,denom)
      short byteCount;
      Ptr textAddr;
      Point numer,denom;
   pascal void MyLine(newPt)
      Point newPt;
   pascal void MyRect(verb,r)
      GrafVerb verb;
      Rect *r;
   pascal void MyRRect(verb,r,ovWd,ovHt)
      GrafVerb verb;
      Rect *r;
      short ovWd,ovHt;
   pascal void MyOval(verb,r)
      GrafVerb verb;
      Rect *r;
   pascal void MyArc(verb,r,startAngle,arcAngle)
      GrafVerb verb;
      Rect *r;
      short startAngle,arcAngle;
   pascal void MyPoly(verb,poly)
      GrafVerb verb;
      PolyHandle poly;
   pascal void MyRgn(verb,rgn)
      GrafVerb verb;
      RgnHandle rgn;
   pascal void MyBits(srcBits,srcRect,dstRect,mode,maskRgn)
      BitMap *srcBits;
      Rect *srcRect,*dstRect;
      short mode;
      RgnHandle maskRgn;
   pascal void MyComment(kind,dataSize,dataHandle)
      short kind,dataSize;
   Handle dataHandle;
pascal short MyTxMeas(byteCount,textAddr,numer,denom,info)
   short byteCount;
   Ptr textAddr;
   Point *numer,*denom;
   FontInfo *info;
pascal void MyGetPic(dataPtr,byteCount)
   Ptr dataPtr;
   short byteCount;
pascal void MyPutPic(dataPtr,byteCount)
   Ptr dataPtr;
   short byteCount;
```

DESCRIPTION

QuickDraw is the Macintosh graphics package.

Warning: User routines MyText and MyLine are not identical to their counterparts StdText and StdLine. Point parameters to MyText and MyLine are passed by value; the corresponding parameters to StdText and StdLine are passed by reference.

For more detailed information see the QuickDraw chapter of *Inside Macintosh.*

NAME

resources—Resource Manager

SYNOPSIS

```
#include <types.h>
#include <resources.h>


/* Resource Attributes */

#define resSysHeap        64        /* set if read into system heap    */
#define resPurgeable      32        /* set if purgeable                */
#define resLocked         16        /* set if locked                   */
#define resProtected       8        /* set if protected                */
#define resPreload         4        /* set if to be preloaded          */
#define resChanged         2        /* set if written to resource file */

/* Error Numbers */

#define resNotFound    (-192)       /* resource not found              */
#define resFNotFound   (-193)       /* resource file not found         */
#define addResFailed   (-194)       /* AddResource failed              */
#define rmvResFailed   (-196)       /* RmveResource failed             */

/* Resource File Attributes */

#define mapReadOnly      128        /* set if file is read-only        */
#define mapCompact        64        /* set to compact file on update   */
#define mapChanged        32        /* set if write map on update      */

/* typedef long ResType;   appears in file TYPES */


/* Initialization */

pascal short InitResources()
pascal void RsrcZoneInit()


/* Opening and Closing Resource Files */

void CreateResFile(fileName)
   char *fileName;
short OpenResFile(fileName)
   char *fileName;
pascal void CloseResFile(refNum)
   short refNum;


/* Checking for Errors */

pascal short ResError()


/* Setting the Current Resource File */

pascal short CurResFile()
pascal short HomeResFile(theResource)
   Handle theResource;
pascal void UseResFile(refNum)
   short refNum;


/*  Getting Resource Types */

pascal short CountTypes()
pascal void GetIndType(theType,index)
   ResType *theType;
   short index;


/* Getting and Displosing of Resources */

pascal void SetResLoad(load)
   Boolean load;
pascal short CountResources(theType)
   ResType theType;
pascal Handle GetIndResource(theType,index)
   ResType theType;
   short index;
pascal Handle GetResource(theType,theID)
   ResType theType;
   short theID;
Handle GetNamedResource(theType,name)
   ResType theType;
   char *name;
pascal void LoadResource(theResource)
   Handle theResource;
pascal void ReleaseResource(theResource)
   Handle theResource;
pascal void DetachResource(theResource)
   Handle theResource;


/* Getting Resource Information */

pascal short UniqueID(theType)
   ResType theType;
void GetResInfo(theResource,theID,theType,name)
   Handle theResource;
   short *theID;
   ResType *theType;
   char *name;
pascal short GetResAttrs(theResource)
   Handle theResource;
pascal long SizeResource(theResource)
   Handle theResource;


/* Modifing Resources */

void SetResInfo(theResource,theID,name)
   Handle theResource;
   short theID;
   char *name;
pascal void SetResAttrs(theResource,attrs)
   Handle theResource;
   short attrs;
pascal void ChangedResource(theResource)
   Handle theResource;
void AddResource(theData,theType,theID,name)
   Handle theData;
   ResType theType;
   short theID;
   char *name;
pascal void RmveResource(theResource)
   Handle theResource;
pascal void UpdateResFile(refNum)
   short refNum;
pascal void WriteResource(theResource)
   Handle theResource;
pascal void SetResPurge(install)
   Boolean install;

/* Advanced Routines */

pascal short GetResFileAttrs(refNum)
   short refNum;
pascal void SetResFileAttrs(refNum,attrs)
   short refNum;
   short attrs;
```

DESCRIPTION

The Resource Manager provides access to Macintosh resource files.

ResType may be specified as a character literal (e.g., 'MENU').

For more detailed information see the Resource Manager chapter of *Inside Macintosh*.

NAME

retrace—Vertical Retrace Manager

SYNOPSIS

```
#include <types.h>
#include <retrace.h>


/* error codes */

#define  noErr           0       /* no error */
#define  qErr           (-1)     /* task entry isn't in the queue */
#define  vTypErr        (-2)     /* qType field isn't vType */


typedef struct VBLTask {
   struct QElem *qLink;          /* next queue entry */
   short         qType;          /* unique id for validity check */
   ProcPtr       vblAddr;        /* address of service routine */
   short         vblCount;       /* count field for timeout */
   short         vblPhase;       /* phase to allow synchronization */
} VBLTask;


OSErr VInstall(vblTaskPtr)
   QElemPtr vblTaskPtr;
OSErr VRemove(vblTaskPtr)
   QElemPtr vblTaskPtr;
struct QHdr *GetVBLQHdr()
```

DESCRIPTION

The Vertical Retrace Manager schedules and performs recurrent tasks during vertical retrace interrupts.

For more detailed information see the Vertical Retrace Manager chapter of *Inside Macintosh.*

NAME

sane—SANE Numerics

SYNOPSIS

```
#include <sane.h>

/* Decimal Representation Constants */

#define  SIGDIGLEN          20     /* significant decimal digits      */
#define  DECSTROUTLEN       80     /* max length for decimal string  */
                                   /*   output                        */

/* Decimal Formatting Styles */

#define  FLOATDECIMAL        0
#define  FIXEDDECIMAL        1

/* Exceptions */

#define  INVALID             1
#define  UNDERFLOW           2
#define  OVERFLOW            4
#define  DIVBYZERO           8
#define  INEXACT            16

/* Ordering Relations */

#define  GREATERTHAN         0
#define  LESSTHAN            1
#define  EQUALTO             2
#define  UNORDERED           3

/* Inquiry Classes */

#define  SNAN                0
#define  QNAN                1
#define  INFINITE            2
#define  ZERONUM             3
#define  NORMALNUM           4
#define  DENORMALNUM         5

/* Rounding Directions */

#define  TONEAREST           0
#define  UPWARD              1
#define  DOWNWARD            2
#define  TOWARDZERO          3

/* Rounding Precisions */

#define  EXTPRECISION        0
#define  DBLPRECISION        1
#define  FLOATPRECISION      2


typedef short exception;      /* sum of INVALID...INEXACT             */
typedef short relop;          /* relational operator                    */

typedef short numclass;       /* inquiry class                          */

typedef short rounddir;       /* rounding direction                     */

typedef short roundpre;       /* rounding precision                     */

typedef short environment;

typedef struct decimal {
   char sgn, unused;          /* sign 0 for +, 1 for -                  */
   short exp;                 /* decimal exponent                       */
   struct {unsigned char length, text[SIGDIGLEN], unused} sig;
                              /* significant digits                     */
} decimal;

typedef struct decform {
   char style, unused;        /* FLOATDECIMAL or FIXEDDECIMAL           */
   short digits;
} decform;

typedef void (*haltvector)();


/* Conversions between Binary and Decimal Records */

void num2dec(f,x,d)           /* d <-- x, according to format f         */
   decform *f;
   extended x;
   decimal *d;
extended dec2num(d)           /* returns d as extended                  */
   decimal *d;

/* Conversions between Decimal Records and ASCII Strings */

void dec2str(f,d,s)           /* s <-- d, according to format f         */
   decform *f;
   decimal *d;
   char *s;
void str2dec(s,ix,d,vp) /* on input ix is starting index into s, on    */
   char *s;             /* output ix is one greater than index of last*/
   short *ix,*vp;       /* character of longest numeric substring;    */
   decimal *d;          /* boolean vp = "s begining at given ix is a  */
                        /* valid numeric string or a valid prefix of  */
                        /* some numeric string"                       */

/* Arithmetic, Auxiliary, and Elementary Functions */

extended fabs(x)              /* absolute value                         */
   extended x;
extended remainder(x,y,quo)   /* IEEE remainder; quo <-- 7 low order    */
   extended x,y;              /*   bits of integer quotient x/y,        */
   short *quo;                /*   -127 <= quo <= 127                   */
extended sqrt(x)              /* square root                            */
   extended x;
extended rint(x)             /* round to integral value              */
   extended x;
extended scalb(n,x)          /* binary scale: x * 2^n;               */
   short n;
   extended x;
extended logb(x)             /* binary log:                          */
   extended x;               /*   binary exponent of normalized x    */
extended copysign(x,y)       /* y with sign of x                     */
   extended x,y;
extended nextfloat(x,y)      /* next float representation after      */
   extended x,y;             /*   (float) x in direction of (float) y*/
extended nextdouble(x,y)     /* next double representation after     */
   extended x,y;             /*  (double) x in direction of (double) y */
extended nextextended(x,y)   /* next extended representation after x */
   extended x,y;             /*   in direction of y                  */
extended log2(x)             /* base-2 log                           */
   extended x;
extended log(x)              /* base-e log                           */
   extended x;
extended log1(x)             /* log(1 + x)                           */
   extended x;
extended exp2(x)             /* base-2 exponential                   */
   extended x;
extended exp(x)              /* base-e exponential                   */
   extended x;
extended exp1(x)             /* exp(x) - 1                           */
   extended x;
extended power(x,y)          /* general exponential: x ^ y           */
   extended x,y;
extended ipower(x,i)         /* integer exponential: x ^ i           */
   extended x;
   short i;
extended compound(r,n)       /* compound: (1 + r) ^ n                */
   extended r,n;
extended annuity(r,n)        /* annuity: (1 - (1 + r) ^ (-n)) / r    */
   extended r,n;
extended tan(x)              /* tangent                              */
   extended x;
extended sin(x)              /* sine                                 */
   extended x;
extended cos(x)              /* cosine                               */
   extended x;
extended atan(x)             /* arctangent                           */
   extended x;
extended randomx(x)          /* returns next random number; updates x; */
   extended *x;              /*   x integral, 1 <= x <= 2^31 - 2     */

/* Inquiry Routines */

numclass classfloat(x)       /* class of (float) x                   */
   extended x;
numclass classdouble(x)      /* class of (double) x                  */
   extended x;
numclass classcomp(x)        /* class of (comp) x                    */
   extended x;
numclass classextended(x)    /* class of x                           */
   extended x;
```

```
long signnum(x)              /* returns 0 for +, 1 for -            */
   extended x;

/* Environment Access Routines */
/*  An exception variable encodes the exceptions
            whose sum is its value */

void setexception(e,s)       /* clrs e flags if s is 0, sets e flags */
   exception e;              /*   otherwise; may cause halt          */
   long s;
long testexception(e)        /* returns 1 if any e flag is set,      */
   exception e;              /*   returns 0 otherwise                */
void sethalt(e,s)            /* disables e halts if s is 0,          */
   exception e;              /*   enables e halts otherwise          */
   long s;
long testhalt(e)             /* returns 1 if any e halt is enabled,  */
   exception e;              /*   returns 0 otherwise                */
void setround(r)             /* sets rounding direction to r         */
   rounddir r;
rounddir getround()          /* returns rounding direction           */
void setprecison(p)          /* sets rounding precision to p         */
   roundpre p;
roundpre getprecsion()       /* returns rounding precision           */
void setenvironment(e)       /* sets environment to e                */
   environment e;
void getenvironment(e)       /* e <-- environment                    */
   environment *e;
void procentry(e)            /* e <-- environment;                   */
   environment *e;           /*   environment <-- IEEE default       */
void procexit(e)             /* temp <--exceptions; environment <-- e; */
   environment e;            /*   signals exceptions in temp         */
haltvector gethaltvector()   /* returns halt vector                  */
void sethaltvector(v)        /* halt vector <-- v                    */
   haltvector v;

/* Comparision Routine */

relop relation(x,y)          /* returns relation such that           */
   extended x,y;             /*   "x relation y" is true             */

/* NaNs and Special Constants */

extended nan(c)              /* returns NaN with code c              */
   unsigned char c;
extended inf()               /* infinity                             */
extended pi()                /* pi                                   */
```

DESCRIPTION

These routines together with Apple's C language fully support the Standard Apple Numeric Environment (SANE). They comprise a scrupulously conforming implementation of extended-precision IEEE Standard 754 floating-point arithmetic.

The Standard Apple Numeric Environment is documented in the *Apple Numerics Manual*.

NAME

scrap—Scrap Manager

SYNOPSIS

```
#include <types.h>
#include <scrap.h>


#define noScrapErr   (-100)   /* desk scrap isn't initialized  */
#define noTypeErr    (-102)   /* no data of the requested type */

typedef struct ScrapStuff {
   long      scrapSize;
   Handle    scrapHandle;
   short     scrapCount;
   short     scrapState;
   StringPtr scrapName;
} ScrapStuff, *PScrapStuff;


/* Getting Desk Scrap Information */

pascal PScrapStuff InfoScrap()

/* Keeping the Desk Scrap on the Disk */

pascal long UnloadScrap()
pascal long LoadScrap()

/* Reading from the Desk Scrap */

pascal long GetScrap(hDest,theType,offset)
   Handle hDest;
   ResType theType;
   long *offset;

/* Writing to the Desk Scrap */

pascal long ZeroScrap()
pascal long PutScrap(length,theType,source)
   long length;
   ResType theType;
   Ptr source;
```

DESCRIPTION

The Scrap Manager provides a mechanism for cutting and pasting between applications and desk accessories.

For more detailed information see the Scrap Manager chapter of *Inside Macintosh.*

NAME

segload—Segment Loader

SYNOPSIS

```
#include <types.h>
#include <segload.h>


/* Message returned by CountAppFiles */

#define  appOpen         0        /* Open the document(s) */
#define  appPrint        1        /* Print the document(s) */


typedef struct AppFile {
   short      vRefNum;            /* volume reference number */
   OSType     fType;              /* file type */
   short      versNum;            /* version number */
   Str255     fName;              /* file name */
} AppFile;


pascal void UnloadSeg(routineAddr)
   Ptr routineAddr;
void CountAppFiles(message,count)
   short *message,*count;
void GetAppFiles(index,theFile)
   short index;
   AppFile *theFile;
void ClrAppFiles(index)
   short index;
void GetAppParms(apName,apRefNum,apParam)
   char *apName;
   short *apRefNum;
   Handle *apParam;
pascal void ExitToShell()
```

DESCRIPTION

The Segment Loader is the part of the Macintosh Operating System that lets you divide your application into several parts and have only some of them in memory at a time. When an application starts up, the Segment Loader also provides it with a list of files to open or print.

For more detailed information see the Segment Loader chapter of *Inside Macintosh.*

NAME

serial—Serial Drivers

SYNOPSIS

```
#include <types.h>
#include <serial.h>


/* Driver reset information */

#define  baud300        380    /*   300 baud */
#define  baud600        189    /*   600 baud */
#define  baud1200        94    /*  1200 baud */
#define  baud1800        62    /*  1800 baud */
#define  baud2400        46    /*  2400 baud */
#define  baud3600        30    /*  3600 baud */
#define  baud4800        22    /*  4800 baud */
#define  baud7200        14    /*  7200 baud */
#define  baud9600        10    /*  9600 baud */
#define  baud19200        4    /* 19200 baud */
#define  baud57600        0    /* 57600 baud */
#define  stop10       16384    /* 1 stop bit */
#define  stop15     (-32768)   /* 1.5 stop bits */
#define  stop20     (-16384)   /* 2 stop bits */
#define  noParity         0    /* no parity */
#define  oddParity     4096    /* odd parity */
#define  evenParity   12288    /* even parity */
#define  data5            0    /* 5 data bits */
#define  data6         2048    /* 6 data bits */
#define  data7         1024    /* 7 data bits */
#define  data8         3072    /* 8 data bits */


/* Masks for errors */

#define  swOverrunErr     1    /* set if software overrun error */
#define  parityErr       16    /* set if parity error */
#define  hwOverrunErr    32    /* set if hardware overrun error */
#define  framingErr      64    /* set if framing error */


/* Masks for changes that cause events to be posted */

#define  ctsEvent        32    /* set if CTS change will cause event */
                               /*  to be posted */
#define  breakEvent     128    /* set if break status change will */
                               /*   cause event to be posted */


/* Indication that an XOFF character was sent */

#define  xOffWasSent   0x80    /* XOFF character was sent */


/* Result codes */

#define  noErr            0    /* no error */
```

```
#define  openErr       ( -23)   /* Open of RAM Serial Driver failed */


typedef enum {
   sPortA,                      /* modem port */
   sPortB                       /* printer port */
} SPortSel;

typedef struct SerShk {
   char          fXOn;          /* XON/XOFF output flow control flag */
   char          fCTS;          /* CTS hardware handshake flag */
   unsigned char xOn;           /* XOn character */
   unsigned char xOff;          /* XOff character */
   char          errs;          /* errors that cause abort */
   char          evts;          /* status changes that cause events */
   char          fInX;          /* XOn/XOff input flow control flag */
   char          null;          /* not used */
} SerShk;

typedef struct SerStaRec {
   char          cumErrs;       /* cumulative errors */
   char          xOffSent;      /* XOff sent as input flow control */
   char          rdPend;        /* read pending flag */
   char          wrPend;        /* write pending flag */
   char          ctsHold;       /* CTS flow control hold flag */
   char          xOffHold;      /* XOff received as output flow control */
} SerStaRec;


/* Opening and Closing the RAM Serial Driver */

OSErr RAMSDOpen(whichPort)
   SPortSel whichPort;
void RAMSDClose(whichPort)
   SPortSel whichPort;


/* Changing Serial Driver Information */

OSErr SerReset(refNum,serConfig)
   short refNum,serConfig;
OSErr SerSetBuf(refNum,serBPtr,serBLen)
   short refNum;
   Ptr serBPtr;
   short serBLen;
OSErr SerHShake(refNum,flags);
   short refNum;
   SerShk *flags;
OSErr SerSetBrk(refNum)
   short refNum;
OSErr SerClrBrk(refNum)
   short refNum;


/* Getting Serial Driver Information */
```

```
OSErr SerGetBuf(refNum,count)
   short refNum;
   long *count;
OSErr SerStatus(refNum,serSta)
   short refNum;
   SerStaRec *serSta;
```

DESCRIPTION

The RAM Serial Driver and the ROM Serial Driver are Macintosh device drivers for handling asynchronous serial communication between a Macintosh application and serial devices.

For more detailed information see the Serial Drivers chapter of *Inside Macintosh.*

NAME

sound—Sound Driver

SYNOPSIS

```
#include <types.h>
#include <sound.h>


/* Mode values for synthesizers */

#define  swMode         (-1)    /* square-wave synthesizer */
#define  ftMode          1      /* four-tone synthesizer */
#define  ffMode          0      /* free-form synthesizer */


/* Free-Form synthesizer */

typedef unsigned char FreeWave[30001];

typedef struct FFSynthRec {
   short       mode;            /* always ffMode */
   Fixed       count;           /* "sizing" factor */
   FreeWave    waveBytes;       /* waveform description */
} FFSynthRec, *FFSynthPtr;


/* Square-Wave synthesizer */

typedef struct Tone {
   short       count;           /* frequency */
   short       amplitude;       /* amplitude, 0-255 */
   short       duration;        /* duration in ticks */
} Tone;

typedef Tone Tones[5001];

typedef struct SWSynthRec {
   short       mode;            /* always swMode */
   Tones       triplets;        /* sounds */
} SWSynthRec, *SWSynthPtr;


/* Four-Tone Synthesizer */

typedef unsigned char Wave[256];

typedef Wave *WavePtr;

typedef struct FTSoundRec {
   short       duration;        /* duration in ticks */
   Fixed       sound1Rate;      /* tone 1 cycle rate */
   long        sound1Phase;     /* tone 1 byte offset */
   Fixed       sound2Rate;      /* tone 2 cycle rate */
   long        sound2Phase;     /* tone 2 byte offset */
   Fixed       sound3Rate;      /* tone 3 cycle rate */
   long        sound3Phase;     /* tone 3 byte offset */
   Fixed       sound4Rate;      /* tone 4 cycle rate */
   long        sound4Phase;     /* tone 4 byte offset */
   WavePtr     sound1Wave;      /* tone 1 wave form */
   WavePtr     sound2Wave;      /* tone 2 wave form */
   WavePtr     sound3Wave;      /* tone 3 wave form */
   WavePtr     sound4Wave;      /* tone 4 wave form */
} FTSoundRec, *FTSndRecPtr;

typedef struct FTSynthRec {
   short       mode;            /* always ftMode */
   FTSndRecPtr sndRec;          /* tones to play */
} FTSynthRec, *FTSynthPtr;


void StartSound(synthRec,numBytes,completionRtn)
   Ptr synthRec;
   long numBytes;
   ProcPtr completionRtn;
void StopSound()
Boolean SoundDone()
void GetSoundVol(level)
   short *level;
void SetSoundVol(level)
   short level
```

DESCRIPTION

The Sound Driver is a Macintosh device driver for handling sound and music generation in a Macintosh application.

For more detailed information see the Sound Driver chapter of *Inside Macintosh.*

NAME

strings—string conversions

SYNOPSIS

```
#include <types.h>
#include <strings.h>

String(0) *c2pstr(s)
   char *s;
char *p2cstr(s)
   String(0) *s;
```

DESCRIPTION

C2pstr converts *s* from a C string to a Pascal string. *P2cstr* converts *s* from a Pascal string to a C string. Both conversions are done in place. For convenience, *c2pstr* and *p2cstr* return *s* as their function result. Both functions will accept nil as their parameter and do nothing.

Pascal strings begin with a length byte. C strings are terminated by a zero byte. The macro *String* is defined in file types.h.

NAME

textedit—TextEdit

SYNOPSIS

```
#include <types.h>
#include <textedit.h>

#define teJustLeft      0
#define teJustCenter    1
#define teJustRight   (-1)

typedef char Chars[32001];
typedef Chars *CharsPtr, **CharsHandle;

typedef struct TERec {
   Rect     destRect;           /* destination rectangle */
   Rect     viewRect;           /* view rectangle */
   Rect     selRect;            /* select rectangle */
   short    lineHeight;         /* current font lineheight */
   short    fontAscent;         /* current font ascent */
   Point    selPoint;           /* selection point (mouseLoc) */
   short    selStart;           /* selection start */
   short    selEnd;             /* selection end */
   short    active;             /* != 0 if active */
   ProcPtr  wordBreak;          /* word break routine */
   ProcPtr  clikLoop;           /* click loop routine */
   long     clickTime;          /* time of first click */
   short    clickLoc;           /* char. location of click */
   long     caretTime;          /* time for next caret blink */
   short    caretState;         /* on/active booleans */
   short    just;               /* fill style */
   short    teLength;           /* length of text below */
   Handle   hText;              /* handle to actual text */
   short    recalBack;          /* != 0 if recal in background */
   short    recalLines;         /* line being recalulated */
   short    clikStuff;          /* click stuff (internal) */
   short    crOnly;             /* set to -1 if CR Line breaks only */
   short    txFont;             /* text Font */
   Style    txFace;             /* text Face */
   short    txMode;             /* text Mode */
   short    txSize;             /* text Size */
   struct GrafPort *inPort;     /* GrafPort */
   ProcPtr  highHook;           /* highlighting hook */
   ProcPtr  caretHook;          /* highlighting hook */
   short    nLines;             /* number of lines */
   short    lineStarts[16001];  /* line starts */
} TERec, *TEPtr, **TEHandle;


/* Initialization and Allocation */

pascal void TEInit()
pascal TEHandle TENew(destRect,viewRect)
   Rect *destRect, *viewRect;
pascal void TEDispose(h)
   TEHandle h;
```

```
/* Accessing Text */

pascal void TESetText(text,length,hTE)
   Ptr text;
   long length;
   TEHandle hTE;
pascal CharsHandle TEGetText(hTE)
   TEHandle hTE;

/* Insertion Point and Selection Range */

pascal void TEIdle(hTE)
   TEHandle hTE;
void TEClick(pt,extend,hTE)
   Point *pt;
   Boolean extend;
   TEHandle hTE;
pascal void TESetSelect(selStart,selEnd,hTE)
   long selStart;
   long selEnd;
   TEHandle hTE;
pascal void TEActivate(hTE)
   TEHandle hTE;
pascal void TEDeactivate(hTE)
   TEHandle hTE;

/* Editing */

pascal void TEKey(key,hTE)
   short key;
   TEHandle hTE;
pascal void TECut(hTE)
   TEHandle hTE;
pascal void TECopy(hTE)
   TEHandle hTE;
pascal void TEPaste(hTE)
   TEHandle hTE;
pascal void TEDelete(hTE)
   TEHandle hTE;
pascal void TEInsert(text,length,hTE)
   Ptr text;
   long length;
   TEHandle hTE;

/* Text Display and Scrolling */

pascal void TESetJust(just,hTE)
   short just;
   TEHandle hTE;
pascal void TEUpdate(rUpdate,hTE)
   Rect *rUpdate;
   TEHandle hTE;
pascal void TextBox(text,length,box,just)
   Ptr text;
   long length;
   Rect *box;
   short just;
```

```
pascal void TEScroll(dh,dv,hTE)
   short dh;
   short dv;
   TEHandle hTE;

/* Scrap Information */

OSErr TEFromScrap()
OSErr TEToScrap()
Handle TEScrapHandle()
long TEGetScrapLen()
void TESetScrapLen(length)
   long length;

/* Advanced Routines */

void SetWordBreak(wBrkProc,hTE)
   ProcPtr wBrkProc;
   TEHandle hTE;
void SetClikLoop(clikProc,hTE)
   ProcPtr clikProc;
   TEHandle hTE;
pascal void TECalText(hTE)
   TEHandle hTE;
```

USER ROUTINES

```
Pascal Boolean MyWordBreak(text,charPos)
   Ptr text;
   short charPos;
Pascal Boolean MyClikLoop()
```

DESCRIPTION

The TextEdit package provides basic text formatting and editing.

For more detailed information see the TextEdit chapter of *Inside Macintosh.*

NOTE

The user routines *highHook* and *caretHook* are called with register conventions and therefore can't be C routines.

NAME
toolutils—Toolbox Utilities

SYNOPSIS
```
#include <types.h>
#include <toolutils.h>


#define sysPatListID    0  /* resource ID of standard pattern list */

#define iBeamCursor     1  /* text selection */
#define crossCursor     2  /* drawing graphics */
#define plusCursor      3  /* cell selection */
#define watchCursor     4  /* indicating long delay */


typedef struct Int64Bit {
   long hiLong;
   long loLong;
} Int64Bit;

typedef struct Cursor *CursPtr, **CursHandle;
typedef struct Pattern *PatPtr, **PatHandle;

/* Fixed-Point Arithmetic */

pascal Fixed FixRatio(numer,denom)
   short numer,denom;
pascal Fixed FixMul(a,b)
   Fixed a,b;
pascal short FixRound(x)
   Fixed x;

/* String Manipulation */

StringHandle NewString(theString)
   char *theString;
void SetString(h,theString)
   StringHandle h;
   char *theString;
pascal StringHandle GetString(stringID)
   short stringID;
void GetIndString(theString,strListID,index)
   char *theString;
   short strListID;
   short index;

/* Byte Manipulation */

pascal long Munger(h,offset,ptr1,length1,ptr2,length2)
   Handle h;
   long offset;
   Ptr ptr1;
   long length1;
   Ptr ptr2;
   long length2;
pascal void PackBits(srcPtr,dstPtr,srcBytes)
```

```
   Ptr *srcPtr;
   Ptr *dstPtr;
   short srcBytes;
pascal void UnpackBits(srcPtr,dstPtr,dstBytes)
   Ptr *srcPtr;
   Ptr *dstPtr;
   short dstBytes;

/* Bit Manipulation */

pascal Boolean BitTst(bytePtr,bitNum)
   Ptr bytePtr;
   long bitNum;
pascal void BitSet(bytePtr,bitNum)
   Ptr bytePtr;
   long bitNum;
pascal void BitClr(bytePtr,bitNum)
   Ptr bytePtr;
   long bitNum;

/* Logical Operations */

pascal long BitAnd(value1,value2)
   long value1,value2;
pascal long BitOr(value1,value2)
   long value1,value2;
pascal long BitXor(value1,value2)
   long value1,value2;
pascal long BitNot(value)
   long value;
pascal long BitShift(value,count)
   long value;
   short count;

/* Other Operations on Ints */

pascal short HiWord(x)
   long x;
pascal short LoWord(x)
   long x;
pascal void LongMul(a,b,dest)
   long a,b;
   Int64Bit *dest;

/* Graphics Utilities */

void ScreenRes(scrnHRes,scrnVRes)
   short *scrnHRes;
   short *scrnVRes;
pascal Handle GetIcon(iconID)
   short iconID;
pascal void PlotIcon(theRect,theIcon)
   Rect *theRect;
   Handle theIcon;
pascal PatHandle GetPattern(patID)
   short patID;
void GetIndPattern(thePattern,patListID,index)
```

```
   struct Pattern *thePattern;
   short patListID;
   short index;
pascal CursHandle GetCursor(cursorID)
   short cursorID;
void ShieldCursor(shieldRect,offsetPt)
   Rect *shieldRect;
   Point *offsetPt;
pascal struct Picture **GetPicture(picID)
   short picID;

/* Miscellaneous Utilities */

long DeltaPoint(ptA,ptB)
   Point *ptA;
   Point *ptB;
pascal Fixed SlopeFromAngle(angle)
   short angle;
pascal short AngleFromSlope(slope)
   Fixed slope;
```

DESCRIPTION

The Toolbox Utilities provide fixed-point arithmetic; string, byte, and bit manipulation; logical operations; and some graphics utilities.

Warning: *NewString* and *GetString* return handles to Pascal strings. *SetString* takes a C string as its parameter and converts it to a Pascal string before storing it in the memory location referred to by the string handle.

For more detailed information see the ToolBox Utilities chapter of *Inside Macintosh.*

NAME
types—common defines and types

SYNOPSIS
```
#include <types.h>

#define nil 0
#define NULL 0

typedef enum {false,true} Boolean;
typedef char *Ptr;
typedef Ptr *Handle;
typedef long (*ProcPtr)();
typedef ProcPtr *ProcHandle;
typedef long Fixed;
typedef unsigned long ResType;
typedef long OSType;
typedef short OSErr;
typedef short Style;
typedef struct Point {
   short    v;
   short    h;
} Point;
typedef struct Rect {
   short    top;
   short    left;
   short    bottom;
   short    right;
} Rect;

#define String(size) struct {\
   unsigned char length; unsigned char text[size];}
typedef String(255) Str255, *StringPtr, **StringHandle;
```

DESCRIPTION
These defines and types are shared by several Macintosh libraries.

The define *String* approximates Pascal strings. It creates a struct, not an array. Remember to use & when passing structs as parameters.

NAME

windows—Window Manager

SYNOPSIS

```
#include <types.h>
#include <quickdraw.h>
#include <windows.h>


/* Window Definition Procedure IDs */

#define documentProc     0
#define dBoxProc         1
#define plainDBox        2
#define altDBoxProc      3
#define noGrowDocProc    4
#define rDocProc        16

/* Types of Windows */

#define dialogKind       2
#define userKind         8

/* FindWindow Result Codes */

#define inDesk           0
#define inMenuBar        1
#define inSysWindow      2
#define inContent        3
#define inDrag           4
#define inGrow           5
#define inGoAway         6

/* Axis Constraints for DragGrayRgn */

#define noConstraint     0
#define hAxisOnly        1
#define vAxisOnly        2

/* Messages to window definition functions */

#define wDraw            0
#define wHit             1
#define wCalcRgns        2
#define wNew             3
#define wDispose         4
#define wGrow            5
#define wDrawGIcon       6

/* defProc Hit Test Codes */

#define wNoHit           0
#define wInContent       1
#define wInDrag          2
#define wInGrow          3
#define wInGoAway        4
```

```
#define deskPatID      16

typedef GrafPtr WindowPtr;

typedef struct WindowRecord {
   GrafPort       port;
   short          windowKind;
   Boolean        visible;
   Boolean        hilited;
   Boolean        goAwayFlag;
   Boolean        spareFlag;
   RgnHandle      strucRgn;
   RgnHandle      contRgn;
   RgnHandle      updateRgn;
   ProcHandle     windowDefProc;
   Handle         dataHandle;
   StringHandle   titleHandle;
   short          titleWidth;
   struct ControlRecord **controlList;
   struct WindowRecord *nextWindow;
   PicHandle      windowPic;
   long           refCon;
} WindowRecord, *WindowPeek;


/* Initialization and Allocation */

pascal void InitWindows()
pascal void GetWMgrPort(wPort)
   GrafPtr *wPort;
WindowPtr NewWindow(wStorage,boundsRect,title,visible,procID,behind,
      goAwayFlag,refCon)
   Ptr wStorage;
   Rect *boundsRect;
   char *title;
   Boolean visible;
   short procID;
   WindowPtr behind;
   Boolean goAwayFlag;
   long refCon;
pascal WindowPtr GetNewWindow(windowID,wStorage,behind)
   short windowID;
   Ptr wStorage;
   WindowPtr behind;
pascal void CloseWindow(theWindow)
   WindowPtr theWindow;
pascal void DisposeWindow(theWindow)
   WindowPtr theWindow;

/* Window Display */

void SetWTitle(theWindow,title)
   WindowPtr theWindow;
   char *title;
void GetWTitle(theWindow,title)
   WindowPtr theWindow;
   char *title;
```

```
pascal void SelectWindow(theWindow)
   WindowPtr theWindow;
pascal void HideWindow(theWindow)
   WindowPtr theWindow;
pascal void ShowWindow(theWindow)
   WindowPtr theWindow;
pascal void ShowHide(theWindow,showFlag)
   WindowPtr theWindow;
   Boolean showFlag;
pascal void HiliteWindow(theWindow,fHiLite)
   WindowPtr theWindow;
   Boolean fHiLite;
pascal void BringToFront(theWindow)
   WindowPtr theWindow;
pascal void SendBehind(theWindow,behindWindow)
   WindowPtr theWindow;
   WindowPtr behindWindow;
pascal WindowPtr FrontWindow()
pascal void DrawGrowIcon(theWindow)
   WindowPtr theWindow;

/* Mouse Location */

short FindWindow(thePt,theWindow)
   Point *thePt;
   WindowPtr *theWindow;
Boolean TrackGoAway(theWindow,thePt)
   WindowPtr theWindow;
   Point *thePt;

/* Window Movement and Sizing */

pascal void MoveWindow(theWindow,hGlobal,vGlobal,front)
   WindowPtr theWindow;
   short hGlobal,vGlobal;
   Boolean front;
void DragWindow(theWindow,startPt,boundsRect)
   WindowPtr theWindow;
   Point *startPt;
   Rect *boundsRect;
long GrowWindow(theWindow,startPt,sizeRect)
   WindowPtr theWindow;
   Point *startPt;
   Rect *sizeRect;
pascal void SizeWindow(theWindow,w,h,fUpdate)
   WindowPtr theWindow;
   short w,h;
   Boolean fUpdate;

/* Update Region Maintenance */

pascal void InvalRect(badRect)
   Rect *badRect;
pascal void InvalRgn(badRgn)
   RgnHandle badRgn;
pascal void ValidRect(goodRect)
   Rect *goodRect;
```

```
pascal void ValidRgn(goodRgn)
   RgnHandle goodRgn;
pascal void BeginUpdate(theWindow)
   WindowPtr theWindow;
pascal void EndUpdate(theWindow)
   WindowPtr theWindow;

/* Miscellaneous Utilities */

pascal void SetWRefCon(theWindow,data)
   WindowPtr theWindow;
   long data;
pascal long GetWRefCon(theWindow)
   WindowPtr theWindow;
pascal void SetWindowPic(theWindow,pic)
   WindowPtr theWindow;
   PicHandle pic;
pascal PicHandle GetWindowPic(theWindow)
   WindowPtr theWindow;
long PinRect(theRect,thePt)
   Rect *theRect;
   Point *thePt;
long DragGrayRgn(theRgn,startPt,limitRect,slopRect,axis,actionProc)
   RgnHandle theRgn;
   Point *startPt;
   Rect *limitRect;
   Rect *slopRect;
   short axis;
   ProcPtr actionProc;

/* Low-Level Routines */

pascal Boolean CheckUpdate(theEvent)
   struct EventRecord *theEvent;
pascal void ClipAbove(window)
   WindowPeek window;
pascal void SaveOld(window)
   WindowPeek window;
pascal void DrawNew(window,update)
   WindowPeek window;
   Boolean update;
pascal void PaintOne(window,clobberedRgn)
   WindowPeek window;
   RgnHandle clobberedRgn;
pascal void PaintBehind(startWindow,clobberedRgn)
   WindowPeek startWindow;
   RgnHandle clobberedRgn;
pascal void CalcVis(window)
   WindowPeek window;
pascal void CalcVisBehind(startWindow,clobberedRgn)
   WindowPeek startWindow;
   RgnHandle clobberedRgn;
```

USER ROUTINES

```
pascal MyAction()
pascal long MyWindow(varCode,theWindow,message,param)
   short varCode;
   WindowPtr theWindow;
   short message;
   long param;
```

DESCRIPTION

The Window Manager provides routines for creating and manipulating windows.

For more detailed information see the Window Manager chapter of *Inside Macintosh.*

8. Library Index

The Library Index contains an index entry for all the defines, types, enumeration literals, global variables, and functions defined in the Standard C Library section and the Macintosh Interface Libraries section. These sections are organized alphabetically by library header.

Column 1 contains an alphabetical list of the index entries.

Column 2 specifies the type of declaration (for example, literal) for the index entry.

Column 3 contains the library header under which documentation for the index entry can be found in Section 6 or 7. If column 3 contains "(C)" following the library header—for example, abs(C)—look in the Standard C Library section. Otherwise, look in the Macintosh Interface Libraries section.

Identifier	Type	Library
abbrevDate	literal	packages
abortErr	define	devices
abs	function	abs(C)
acos	function	trig(C)
activateEvt	define	events
activeFlag	define	events
activMask	define	events
AddDrive	function	files
AddPt	function	quickdraw
addResFailed	define	resources
AddResMenu	function	menus
AddResource	function	resources
Alert	function	dialogs
AlertTemplate	type	dialogs
AlertTHndl	type	dialogs
AlertTPtr	type	dialogs
Allocate	function	files
alphaLock	define	events
altDBoxProc	define	windows
AngleFromSlope	function	toolutils
annuity	function	sane
app1Evt	define	events
app1Mask	define	events
app2Evt	define	events
app2Mask	define	events
app3Evt	define	events
app3Mask	define	events
app4Evt	define	events
app4Mask	define	events
AppendMenu	function	menus
AppFile	type	segload
appleMark	define	fonts
applFont	define	fonts
ApplicZone	function	memory
appOpen	define	segload
appPrint	define	segload
asin	function	trig(C)
atan	function	sane
atan	function	trig(C)
atan2	function	trig(C)
athens	define	fonts
atof	function	atof(C)
atoi	function	atoi(C)
autoKey	define	events
autoKeyMask	define	events
autoTrack	define	controls
BackColor	function	quickdraw
BackPat	function	quickdraw
badBtSlpErr	define	disks
badCksmErr	define	disks
badDBtSlp	define	disks
badDCksum	define	disks
badMDBErr	define	files
badUnitErr	define	devices
baud1200	define	serial
baud1800	define	serial
baud19200	define	serial
baud2400	define	serial
baud300	define	serial
baud3600	define	serial
baud4800	define	serial
baud57600	define	serial
baud600	define	serial
baud7200	define	serial
baud9600	define	serial
bdConv	define	packages
bDevCItoh	define	printing
bDevLaser	define	printing
bdNamErr	define	files
bDraftLoop	define	printing
BeginUpdate	function	windows
BitAnd	function	toolutils
BitClr	function	toolutils
BitMap	type	quickdraw
BitNot	function	toolutils
BitOr	function	toolutils
Bits16	type	quickdraw
BitSet	function	toolutils
BitShift	function	toolutils
BitTst	function	toolutils
BitXor	function	toolutils
blackBit	define	quickdraw
blackColor	define	quickdraw
BlockMove	function	memory
blueBit	define	quickdraw
blueColor	define	quickdraw
bold	define	quickdraw
Boolean	type	types
breakEvent	define	serial
BringToFront	function	windows
bSpoolLoop	define	printing
btnCtrl	define	dialogs
btnState	define	events
BUFSIZ	define	setbuf(C)
Button	function	events
c2pstr	function	strings
cairo	define	fonts
calcCRgns	define	controls
CalcMenuSize	function	menus
CalcVis	function	windows
CalcVisBehind	function	windows
calloc	function	malloc(C)
cancel	define	dialogs
cantStepErr	define	disks
CautionAlert	function	dialogs
ceil	function	floor(C)
century	define	packages
ChangedResource	function	resources
char	type	quickdraw
char	type	sound
charCodeMask	define	events
Chars	type	textedit
CharsHandle	type	textedit
CharsPtr	type	textedit
CharWidth	function	quickdraw
checkBoxProc	define	controls
CheckItem	function	menus
checkMark	define	fonts
CheckUpdate	function	windows
chkCtrl	define	dialogs
classcomp	function	sane
classdouble	function	sane
classextended	function	sane
classfloat	function	sane
clearerr	macro	ferror(C)

ClearMenuBar	function	menus
ClipAbove	function	windows
ClipRect	function	quickdraw
clkRdErr	define	osutils
clkWrErr	define	osutils
close	function	close(C)
CloseDeskAcc	function	desk
CloseDialog	function	dialogs
CloseDriver	function	devices
ClosePicture	function	quickdraw
ClosePoly	function	quickdraw
ClosePort	function	quickdraw
CloseResFile	function	resources
CloseRgn	function	quickdraw
CloseWindow	function	windows
ClrAppFiles	function	segload
cmdKey	define	events
CntrlParam	type	devices
ColorBit	function	quickdraw
commandMark	define	fonts
CompactMem	function	memory
compound	function	sane
condense	define	quickdraw
Control	function	devices
controlErr	define	devices
ControlHandle	type	controls
ControlPtr	type	controls
ControlRecord	type	controls
CopyBits	function	quickdraw
CopyRgn	function	quickdraw
copysign	function	sane
cos	function	sane
cos	function	trig(C)
cosh	function	cosh(C)
CouldAlert	function	dialogs
CouldDialog	function	dialogs
CountAppFiles	function	segload
CountMItems	function	menus
CountResources	function	resources
CountTypes	function	resources
courier	define	fonts
creat	function	creat(C)
Create	function	files
CreateResFile	function	resources
crossCursor	define	toolutils
ctnIcon	define	dialogs
ctrlItem	define	dialogs
ctsEvent	define	serial
CurResFile	function	resources
currLeadingZ	define	packages
currNegSym	define	packages
currSymLead	define	packages
currTrailingZ	define	packages
CursHandle	type	toolutils
Cursor	type	quickdraw
Cursor	type	toolutils
CursPtr	type	toolutils
cyanBit	define	quickdraw
cyanColor	define	quickdraw
data5	define	serial
data6	define	serial
data7	define	serial
data8	define	serial
dataVerErr	define	disks
Date2Secs	function	osutils
DateForm	type	packages
DateTimeRec	type	osutils
dayLdingZ	define	packages
DBLPRECISION	define	sane
dBoxProc	define	windows
DCtlEntry	type	devices
DCtlHandle	type	devices
DCtlPtr	type	devices
dec2num	function	sane
dec2str	function	sane
decform	type	sane
decimal	type	sane
DECSTROUTLEN	define	sane
Delay	function	osutils
DeleteMenu	function	menus
DeltaPoint	function	toolutils
DENORMALNUM	define	sane
Dequeue	function	osutils
deskPatID	define	windows
DetachResource	function	resources
dialogKind	define	windows
DialogPeek	type	dialogs
DialogPtr	type	dialogs
DialogRecord	type	dialogs
DialogSelect	function	dialogs
DialogTemplate	type	dialogs
DialogTHndl	type	dialogs
DialogTPtr	type	dialogs
diamondMark	define	fonts
DIBadMount	function	packages
DiffRgn	function	quickdraw
DIFormat	function	packages
DILoad	function	packages
dInstErr	define	devices
dirFulErr	define	files
DisableItem	function	menus
DiskEject	function	disks
diskEvt	define	events
diskMask	define	events
dispCntl	define	controls
DisposDialog	function	dialogs
DisposeControl	function	controls
DisposeMenu	function	menus
DisposeRgn	function	quickdraw
DisposeWindow	function	windows
DisposHandle	function	memory
DisposPtr	function	memory
DIUnLoad	function	packages
DIVBYZERO	define	sane
DIVerify	function	packages
DIZero	function	packages
DlgCopy	function	dialogs
DlgCut	function	dialogs
DlgDelete	function	dialogs
DlgPaste	function	dialogs
dmy	define	packages
documentProc	define	windows
DOWNWARD	define	sane
dragCntl	define	controls
DragControl	function	controls
DragGrayRgn	function	windows

feedOther	literal	printing
feof	macro	ferror(C)
ferror	macro	ferror(C)
fflush	function	fclose(C)
ffMode	define	sound
FFSynthPtr	type	sound
FFSynthRec	type	sound
fgetc	function	getc(C)
fgets	function	gets(C)
fHasBundle	define	files
FILE	define	stdio(C)
fileno	macro	ferror(C)
FileParam	type	files
fill	literal	quickdraw
FillArc	function	quickdraw
FillOval	function	quickdraw
FillPoly	function	quickdraw
FillRect	function	quickdraw
FillRgn	function	quickdraw
FillRoundRect	function	quickdraw
FindControl	function	controls
FindWindow	function	windows
FInfo	type	files
FInitQueue	function	files
fInvisible	define	files
firstDskErr	define	disks
Fixed	type	types
FIXEDDECIMAL	define	sane
fixedFont	define	fonts
FixMul	function	toolutils
FixRatio	function	toolutils
FixRound	function	toolutils
FlashMenuBar	function	menus
fLckdErr	define	files
FLOATDECIMAL	define	sane
FLOATPRECISION	define	sane
floor	function	floor(C)
flPoint	define	packages
FlushEvents	function	osevents
FlushVol	function	files
FMInput	type	fonts
fmod	function	floor(C)
FMOutPtr	type	fonts
FMOutput	type	fonts
fnfErr	define	files
fnOpnErr	define	files
FontInfo	type	quickdraw
FontRec	type	fonts
fontWid	define	fonts
fopen	function	fopen(C)
ForeColor	function	quickdraw
fprintf	function	printf(C)
fputc	function	putc(C)
fputs	function	puts(C)
frame	literal	quickdraw
FrameArc	function	quickdraw
FrameOval	function	quickdraw
FramePoly	function	quickdraw
FrameRect	function	quickdraw
FrameRgn	function	quickdraw
FrameRoundRect	function	quickdraw
framingErr	define	serial
fread	function	fread(C)
free	function	malloc(C)
FreeAlert	function	dialogs
FreeDialog	function	dialogs
FreeMem	function	memory
FreeWave	type	sound
freopen	function	fopen(C)
frexp	function	frexp(C)
FrontWindow	function	windows
fsAtMark	define	files
fscanf	function	scanf(C)
FSClose	function	files
fsCurPerm	define	files
FSDelete	function	files
fseek	function	fseek(C)
fsFromLEOF	define	files
fsFromMark	define	files
fsFromStart	define	files
FSOpen	function	files
fsQType	literal	osutils
fsRdPerm	define	files
fsRdWrPerm	define	files
FSRead	function	files
fsRnErr	define	files
FSWrite	function	files
fsWrPerm	define	files
ftell	function	fseek(C)
ftMode	define	sound
fTrash	define	files
FTSndRecPtr	type	sound
FTSoundRec	type	sound
FTSynthPtr	type	sound
FTSynthRec	type	sound
fwrite	function	fread(C)
geneva	define	fonts
GetAlrtStage	function	dialogs
GetAppFiles	function	segload
GetApplLimit	function	memory
GetAppParms	function	segload
getc	macro	getc(C)
getCancel	define	packages
GetCaretTime	function	events
getchar	function	getc(C)
GetClip	function	quickdraw
GetCRefCon	function	controls
GetCTitle	function	controls
GetCtlAction	function	controls
GetCtlMax	function	controls
GetCtlMin	function	controls
GetCtlValue	function	controls
GetCursor	function	toolutils
GetDateTime	function	osutils
GetDblTime	function	events
GetDCtlEntry	function	devices
GetDItem	function	dialogs
getDlgID	define	packages
getDrive	define	packages
GetDrvQHdr	function	files
getEject	define	packages
getenvironment	function	sane
GetEOF	function	files
GetEvQHdr	function	osevents
GetFInfo	function	files
GetFNum	function	fonts

GetFontInfo	function	quickdraw
GetFontName	function	fonts
GetFPos	function	files
GetFSQHdr	function	files
gethaltvector	function	sane
GetHandleSize	function	memory
GetIcon	function	toolutils
GetIndPattern	function	toolutils
GetIndResource	function	resources
GetIndString	function	toolutils
GetIndType	function	resources
GetItem	function	menus
GetItemIcon	function	menus
GetItemMark	function	menus
GetItemStyle	function	menus
GetIText	function	dialogs
GetKeys	function	events
GetMenu	function	menus
GetMenuBar	function	menus
GetMHandle	function	menus
GetMouse	function	events
GetNamedResource	function	resources
GetNewControl	function	controls
GetNewDialog	function	dialogs
GetNewMBar	function	menus
GetNewWindow	function	windows
GetNextEvent	function	events
getNmList	define	packages
getOpen	define	packages
GetOSEvent	function	osevents
GetPattern	function	toolutils
GetPen	function	quickdraw
GetPenState	function	quickdraw
GetPicture	function	toolutils
GetPixel	function	quickdraw
GetPort	function	quickdraw
getprecsion	function	sane
GetPtrSize	function	memory
GetResAttrs	function	resources
GetResFileAttrs	function	resources
GetResInfo	function	resources
GetResource	function	resources
getround	function	sane
gets	function	gets(C)
GetScrap	function	scrap
getScroll	define	packages
GetSoundVol	function	sound
GetString	function	toolutils
GetSysPPtr	function	osutils
GetTime	function	osutils
GetTrapAddress	function	osutils
GetVBLQHdr	function	retrace
GetVCBQHdr	function	files
GetVInfo	function	files
GetVol	function	files
GetVRefNum	function	files
getw	function	getc(C)
GetWindowPic	function	windows
GetWMgrPort	function	windows
GetWRefCon	function	windows
GetWTitle	function	windows
GetZone	function	memory
GlobalToLocal	function	quickdraw
GrafDevice	function	quickdraw
GrafPort	type	quickdraw
GrafPtr	type	quickdraw
GrafVerb	type	quickdraw
GREATERTHAN	define	sane
greenBit	define	quickdraw
greenColor	define	quickdraw
GrowWindow	function	windows
GZCritical	function	memory
GZSaveHnd	function	memory
haltvector	type	sane
HandAndHand	function	osutils
Handle	type	types
HandleZone	function	memory
HandToHand	function	osutils
hAxisOnly	define	controls
hAxisOnly	define	windows
helvetica	define	fonts
HideControl	function	controls
HideCursor	function	quickdraw
HidePen	function	quickdraw
HideWindow	function	windows
HiliteControl	function	controls
HiliteMenu	function	menus
HiliteWindow	function	windows
HiWord	function	toolutils
HLock	function	memory
HNoPurge	function	memory
HomeResFile	function	resources
HPurge	function	memory
hrLeadingZ	define	packages
HUnlock	function	memory
hwOverrunErr	define	serial
hypot	function	hypot(C)
iBeamCursor	define	toolutils
iconItem	define	dialogs
iIOAbort	define	printing
iMemFullErr	define	printing
inButton	define	controls
inCheckbox	define	controls
inContent	define	windows
inDesk	define	windows
inDownButton	define	controls
inDrag	define	windows
INEXACT	define	sane
inf	function	sane
INFINITE	define	sane
InfoScrap	function	scrap
inGoAway	define	windows
inGrow	define	windows
InitAllPacks	function	packages
InitApplZone	function	memory
initCntl	define	controls
InitCursor	function	quickdraw
InitDialogs	function	dialogs
InitFonts	function	fonts
InitGraf	function	quickdraw
initIWMErr	define	disks
InitMenus	function	menus
InitPack	function	packages
InitPort	function	quickdraw
InitResources	function	resources
InitUtil	function	osutils

InitWindows	function	windows
InitZone	function	memory
inMenuBar	define	windows
inPageDown	define	controls
inPageUp	define	controls
InsertMenu	function	menus
InsertResMenu	function	menus
InsetRect	function	quickdraw
InsetRgn	function	quickdraw
inSysWindow	define	windows
Int64Bit	type	toolutils
inThumb	define	controls
Intl0Hndl	type	packages
Intl0Ptr	type	packages
Intl0Rec	type	packages
Intl1Hndl	type	packages
Intl1Ptr	type	packages
Intl1Rec	type	packages
intUtil	define	packages
inUpButton	define	controls
INVALID	define	sane
InvalRect	function	windows
InvalRgn	function	windows
inverseBit	define	quickdraw
invert	literal	quickdraw
InvertArc	function	quickdraw
InvertOval	function	quickdraw
InvertPoly	function	quickdraw
InvertRect	function	quickdraw
InvertRgn	function	quickdraw
InvertRoundRect	function	quickdraw
ioctl	function	ioctl(C)
ioErr	define	files
IOParam	type	files
ioQType	literal	osutils
iPFMaxPgs	define	printing
ipower	function	sane
iPrAbort	define	printing
iPrBitsCtl	define	printing
iPrDevCtl	define	printing
iPrDrvrRef	define	printing
iPrIOCtl	define	printing
iPrPgFract	define	printing
iPrSavPFil	define	printing
isalnum	macro	ctype(C)
isalpha	macro	ctype(C)
isascci	macro	ctype(C)
iscntrl	macro	ctype(C)
IsDialogEvent	function	dialogs
isdigit	macro	ctype(C)
isgraph	macro	ctype(C)
islower	macro	ctype(C)
isprint	macro	ctype(C)
ispunct	macro	ctype(C)
isspace	macro	ctype(C)
isupper	macro	ctype(C)
isxdigit	macro	ctype(C)
italic	define	quickdraw
itemDisable	define	dialogs
IUCompString	function	packages
IUDatePString	function	packages
IUDateString	function	packages
IUEqualString	function	packages
IUGetIntl	function	packages
IUMagIDString	function	packages
IUMagString	function	packages
IUMetric	function	packages
IUSetIntl	function	packages
IUTimePString	function	packages
IUTimeString	function	packages
kanji	define	fonts
keyCodeMask	define	events
keyDown	define	events
keyDownMask	define	events
KeyMap	type	events
keyUp	define	events
keyUpMask	define	events
KillControls	function	controls
KillIO	function	devices
KillPicture	function	quickdraw
KillPoly	function	quickdraw
lastDskErr	define	disks
ldexp	function	frexp(C)
LESSTHAN	define	sane
Line	function	quickdraw
LineTo	function	quickdraw
LoadResource	function	resources
LoadScrap	function	scrap
LocalToGlobal	function	quickdraw
log	function	exp(C)
log	function	sane
log1	function	sane
log10	function	exp(C)
log2	function	sane
logb	function	sane
london	define	fonts
long	type	types
longDate	literal	packages
LongMul	function	toolutils
losAngeles	define	fonts
LoWord	function	toolutils
lPaintBits	define	printing
lPrLFSixth	define	printing
lPrLineFeed	define	printing
lPrPageEnd	define	printing
lPrReset	define	printing
lScreenBits	define	printing
lseek	function	lseek(C)
macMachine	define	osutils
macXLMachine	define	osutils
magentaBit	define	quickdraw
magentaColor	define	quickdraw
malloc	function	malloc(C)
mapChanged	define	resources
mapCompact	define	resources
MapPoly	function	quickdraw
MapPt	function	quickdraw
mapReadOnly	define	resources
MapRect	function	quickdraw
MapRgn	function	quickdraw
MaxApplZone	function	memory
MaxMem	function	memory
maxSize	define	memory
mChooseMsg	define	menus
mDownMask	define	events
mDrawMsg	define	menus

mdy	define	packages
memccpy	function	memory(C)
memchr	function	memory(C)
memcmp	function	memory(C)
memcpy	function	memory(C)
MemError	function	memory
memFullErr	define	memory
memFullErr	define	osutils
memLockedErr	define	memory
memPurErr	define	memory
memset	function	memory(C)
memWZErr	define	memory
memWZErr	define	osutils
MenuHandle	type	menus
MenuInfo	type	menus
MenuKey	function	menus
MenuPtr	type	menus
MenuSelect	function	menus
minLeadingZ	define	packages
mntLdingZ	define	packages
ModalDialog	function	dialogs
modf	function	frexp(C)
monaco	define	fonts
MoreMasters	function	memory
mouseDown	define	events
mouseUp	define	events
Move	function	quickdraw
MoveControl	function	controls
MoveHHi	function	memory
MovePortTo	function	quickdraw
MoveTo	function	quickdraw
MoveWindow	function	windows
mSizeMsg	define	menus
Munger	function	toolutils
mUpMask	define	events
MyAction	function	controls
MyAction	function	windows
MyArc	function	quickdraw
MyBits	function	quickdraw
MyClikLoop	function	textedit
MyComment	function	quickdraw
MyControl	function	controls
MyDlg	function	packages
MyFileFilter	function	packages
MyFilter	function	dialogs
MyGetPic	function	quickdraw
MyGrowZone	function	memory
MyItem	function	dialogs
MyLine	function	quickdraw
MyMenu	function	menus
MyOval	function	quickdraw
MyPoly	function	quickdraw
MyPutPic	function	quickdraw
MyRect	function	quickdraw
MyRgn	function	quickdraw
MyRRect	function	quickdraw
MySound	function	dialogs
MyText	function	quickdraw
MyTxMeas	function	quickdraw
MyWindow	function	windows
MyWordBreak	function	textedit
nan	function	sane
networkEvt	define	events

networkMask	define	events
NewControl	function	controls
NewDialog	function	dialogs
NewHandle	function	memory
NewMenu	function	menus
NewPtr	function	memory
NewRgn	function	quickdraw
NewString	function	toolutils
NewWindow	function	windows
newYork	define	fonts
nextdouble	function	sane
nextextended	function	sane
nextfloat	function	sane
nil	define	types
nilHandleErr	define	memory
nilHandleErr	define	osutils
noAdrMkErr	define	disks
noConstraint	define	controls
noConstraint	define	windows
noDriveErr	define	disks
noDtaMkErr	define	disks
noErr	define	devices
noErr	define	disks
noErr	define	files
noErr	define	memory
noErr	define	osutils
noErr	define	printing
noErr	define	retrace
noErr	define	serial
noGrowDocProc	define	windows
noMacDskErr	define	files
noMark	define	menus
noNybErr	define	disks
noParity	define	serial
normal	define	quickdraw
normalBit	define	quickdraw
NORMALNUM	define	sane
noScrapErr	define	scrap
NoteAlert	function	dialogs
noteIcon	define	dialogs
notOpenErr	define	devices
notPatBic	define	quickdraw
notPatCopy	define	quickdraw
notPatOr	define	quickdraw
notPatXor	define	quickdraw
notSrcBic	define	quickdraw
notSrcCopy	define	quickdraw
notSrcOr	define	quickdraw
notSrcXor	define	quickdraw
noTypeErr	define	scrap
nsDrvErr	define	disks
nsDrvErr	define	files
nsvErr	define	files
NULL	define	stdio(C)
NULL	define	types
nullEvent	define	events
num2dec	function	sane
numclass	type	sane
NumToString	function	packages
O_APPEND	define	open(C)
ObscureCursor	function	quickdraw
O_CREAT	define	open(C)
oddParity	define	serial

O_EXCL	define	open(C)
offLinErr	define	disks
OffsetPoly	function	quickdraw
OffsetRect	function	quickdraw
OffsetRgn	function	quickdraw
ok	define	dialogs
open	function	open(C)
OpenDeskAcc	function	desk
OpenDriver	function	devices
openErr	define	devices
openErr	define	serial
OpenPicture	function	quickdraw
OpenPoly	function	quickdraw
OpenPort	function	quickdraw
OpenResFile	function	resources
OpenRF	function	files
OpenRgn	function	quickdraw
optionKey	define	events
opWrErr	define	files
O_RDONLY	define	open(C)
O_RDWR	define	open(C)
OSErr	type	types
OSEventAvail	function	osevents
OSType	type	types
O_TRUNC	define	open(C)
outline	define	quickdraw
OVERFLOW	define	sane
O_WRONLY	define	open(C)
p2cstr	function	strings
PackBits	function	toolutils
paint	literal	quickdraw
PaintArc	function	quickdraw
PaintBehind	function	windows
PaintOne	function	windows
PaintOval	function	quickdraw
PaintPoly	function	quickdraw
PaintRect	function	quickdraw
PaintRgn	function	quickdraw
PaintRoundRect	function	quickdraw
paramErr	define	disks
paramErr	define	files
ParamText	function	dialogs
parityErr	define	serial
patBic	define	quickdraw
patCopy	define	quickdraw
PatHandle	type	toolutils
patOr	define	quickdraw
PatPtr	type	toolutils
Pattern	type	quickdraw
Pattern	type	toolutils
patXor	define	quickdraw
PBAllocate	function	files
PBClose	function	files
PBControl	function	devices
PBCreate	function	files
PBDelete	function	files
PBEject	function	files
PBFlushFile	function	files
PBFlushVol	function	files
PBGetEOF	function	files
PBGetFInfo	function	files
PBGetFPos	function	files
PBGetVInfo	function	files
PBGetVol	function	files
PBKillIO	function	devices
PBMountVol	function	files
PBOffLine	function	files
PBOpen	function	files
PBOpenRF	function	files
PBRead	function	files
PBRename	function	files
PBRstFLock	function	files
PBSetEOF	function	files
PBSetFInfo	function	files
PBSetFLock	function	files
PBSetFPos	function	files
PBSetFVers	function	files
PBSetVol	function	files
PBStatus	function	devices
PBUnmountVol	function	files
PBWrite	function	files
PenMode	function	quickdraw
PenNormal	function	quickdraw
PenPat	function	quickdraw
PenSize	function	quickdraw
PenState	type	quickdraw
permErr	define	files
pi	function	sane
PicComment	function	quickdraw
PicHandle	type	quickdraw
picItem	define	dialogs
picLParen	define	quickdraw
PicPtr	type	quickdraw
picRParen	define	quickdraw
Picture	type	quickdraw
PinRect	function	windows
plainDBox	define	windows
PlotIcon	function	toolutils
plusCursor	define	toolutils
Point	type	types
Polygon	type	quickdraw
PolyHandle	type	quickdraw
PolyPtr	type	quickdraw
PortSize	function	quickdraw
posCntl	define	controls
posErr	define	files
PostEvent	function	osevents
pow	function	exp(C)
power	function	sane
PrClose	function	printing
PrCloseDoc	function	printing
PrClosePage	function	printing
PrCtlCall	function	printing
PrDrvrClose	function	printing
PrDrvrDCE	function	printing
PrDrvrOpen	function	printing
PrDrvrVers	function	printing
PrError	function	printing
prInitErr	define	osutils
PrintDefault	function	printing
printf	function	printf(C)
PrJobDialog	function	printing
PrJobMerge	function	printing
procentry	function	sane
procexit	function	sane
ProcHandle	type	types

ProcPtr	type	types
PrOpen	function	printing
PrOpenDoc	function	printing
PrOpenPage	function	printing
propFont	define	fonts
PrPicFile	function	printing
PrSetError	function	printing
PrStlDialog	function	printing
PrValidate	function	printing
prWrErr	define	osutils
PScrapStuff	type	scrap
Pt2Rect	function	quickdraw
PtInRect	function	quickdraw
PtInRgn	function	quickdraw
Ptr	type	types
PtrAndHand	function	osutils
PtrToHand	function	osutils
PtrToXHand	function	osutils
PtrZone	function	memory
PtToAngle	function	quickdraw
PurgeMem	function	memory
pushButProc	define	controls
putc	macro	putc(C)
putCancel	define	packages
putchar	macro	putc(C)
putDlgID	define	packages
putDrive	define	packages
putEject	define	packages
putName	define	packages
puts	function	puts(C)
putSave	define	packages
PutScrap	function	scrap
putw	function	putc(C)
QDProcs	type	quickdraw
QDProcsPtr	type	quickdraw
QElem	type	osutils
QElemPtr	type	osutils
qErr	define	osutils
qErr	define	retrace
QHdr	type	osutils
QHdrPtr	type	osutils
QNAN	define	sane
QTypes	type	osutils
radCtrl	define	dialogs
radioButProc	define	controls
RAMSDClose	function	serial
RAMSDOpen	function	serial
rand	function	rand(C)
Random	function	quickdraw
randomx	function	sane
rDocProc	define	windows
rdVerify	define	files
read	function	read(C)
ReadDateTime	function	osutils
readErr	define	devices
RealFont	function	fonts
realloc	function	malloc(C)
ReallocHandle	function	memory
RecoverHandle	function	memory
Rect	type	types
RectInRgn	function	quickdraw
RectRgn	function	quickdraw
redBit	define	quickdraw
redColor	define	quickdraw
Region	type	quickdraw
relation	function	sane
ReleaseResource	function	resources
relop	type	sane
remainder	function	sane
Rename	function	files
resChanged	define	resources
resCtrl	define	dialogs
ResError	function	resources
ResetAlrtStage	function	dialogs
resFNotFound	define	resources
resLocked	define	resources
resNotFound	define	resources
resPreload	define	resources
resProtected	define	resources
resPurgeable	define	resources
ResrvMem	function	memory
resSysHeap	define	resources
Restart	function	osutils
ResType	type	types
rewind	function	fseek(C)
rfNumErr	define	files
RgnHandle	type	quickdraw
RgnPtr	type	quickdraw
rint	function	sane
RmveResource	function	resources
rmvResFailed	define	resources
rounddir	type	sane
roundpre	type	sane
RsrcZoneInit	function	resources
RstFLock	function	files
sanFran	define	fonts
SaveOld	function	windows
scalb	function	sane
ScalePt	function	quickdraw
scanBT	literal	printing
scanf	function	scanf(C)
scanLR	literal	printing
scanRL	literal	printing
scanTB	literal	printing
ScrapStuff	type	scrap
ScreenRes	function	toolutils
scrollBarProc	define	controls
ScrollRect	function	quickdraw
secLeadingZ	define	packages
Secs2Date	function	osutils
sectNFErr	define	disks
SectRect	function	quickdraw
SectRgn	function	quickdraw
seekErr	define	disks
SelectWindow	function	windows
SelIText	function	dialogs
SendBehind	function	windows
SerClrBrk	function	serial
SerGetBuf	function	serial
SerHShake	function	serial
SerReset	function	serial
SerSetBrk	function	serial
SerSetBuf	function	serial
SerShk	type	serial
SerStaRec	type	serial
SerStatus	function	serial

SetApplBase	function	memory
SetApplLimit	function	memory
setbuf	function	setbuf(C)
SetClikLoop	function	textedit
SetClip	function	quickdraw
SetCRefCon	function	controls
SetCTitle	function	controls
SetCtlAction	function	controls
SetCtlMax	function	controls
SetCtlMin	function	controls
SetCtlValue	function	controls
SetCursor	function	quickdraw
SetDAFont	function	dialogs
SetDateTime	function	osutils
SetDItem	function	dialogs
SetEmptyRgn	function	quickdraw
setenvironment	function	sane
SetEOF	function	files
SetEventMask	function	osevents
setexception	function	sane
SetFInfo	function	files
SetFLock	function	files
SetFontLock	function	fonts
SetFPos	function	files
SetGrowZone	function	memory
sethalt	function	sane
sethaltvector	function	sane
SetHandleSize	function	memory
SetItem	function	menus
SetItemIcon	function	menus
SetItemMark	function	menus
SetItemStyle	function	menus
SetIText	function	dialogs
SetMenuBar	function	menus
SetMenuFlash	function	menus
SetOrigin	function	quickdraw
SetPenState	function	quickdraw
SetPort	function	quickdraw
SetPortBits	function	quickdraw
setprecison	function	sane
SetPt	function	quickdraw
SetPtrSize	function	memory
SetRect	function	quickdraw
SetRectRgn	function	quickdraw
SetResAttrs	function	resources
SetResFileAttrs	function	resources
SetResInfo	function	resources
SetResLoad	function	resources
SetResPurge	function	resources
setround	function	sane
SetSoundVol	function	sound
SetStdProcs	function	quickdraw
SetString	function	toolutils
SetTagBuffer	function	disks
SetTime	function	osutils
SetTrapAddress	function	osutils
SetVol	function	files
SetWindowPic	function	windows
SetWordBreak	function	textedit
SetWRefCon	function	windows
SetWTitle	function	windows
SetZone	function	memory
SFGetFile	function	packages
SFPGetFile	function	packages
SFPPutFile	function	packages
SFPutFile	function	packages
SFReply	type	packages
SFTypeList	type	packages
shadow	define	quickdraw
ShieldCursor	function	toolutils
shiftKey	define	events
shortDate	literal	packages
ShowControl	function	controls
ShowCursor	function	quickdraw
ShowHide	function	windows
ShowPen	function	quickdraw
ShowWindow	function	windows
SIGDIGLEN	define	sane
signnum	function	sane
sin	function	sane
sin	function	trig(C)
sinh	function	sinh(C)
Size	type	memory
SizeControl	function	controls
SizeResource	function	resources
SizeWindow	function	windows
SlopeFromAngle	function	toolutils
SNAN	define	sane
SoundDone	function	sound
SpaceExtra	function	quickdraw
spdAdjErr	define	disks
sPortA	literal	serial
sPortB	literal	serial
SPortSel	type	serial
sPrDrvr	define	printing
sprintf	function	printf(C)
sqrt	function	exp(C)
sqrt	function	sane
srand	function	rand(C)
srcBic	define	quickdraw
srcCopy	define	quickdraw
srcOr	define	quickdraw
srcXor	define	quickdraw
sscanf	function	scanf(C)
StageList	type	dialogs
StartSound	function	sound
statText	define	dialogs
Status	function	devices
statusErr	define	devices
StdArc	function	quickdraw
StdBits	function	quickdraw
StdComment	function	quickdraw
stderr	define	stdio(C)
stdFile	define	packages
StdGetPic	function	quickdraw
stdin	define	stdio(C)
StdLine	function	quickdraw
stdout	define	stdio(C)
StdOval	function	quickdraw
StdPoly	function	quickdraw
StdPutPic	function	quickdraw
StdRect	function	quickdraw
StdRgn	function	quickdraw
StdRRect	function	quickdraw
StdText	function	quickdraw
StdTxMeas	function	quickdraw

StillDown	function	events
stop10	define	serial
stop15	define	serial
stop20	define	serial
StopAlert	function	dialogs
stopIcon	define	dialogs
StopSound	function	sound
Str255	type	types
str2dec	function	sane
strcat	function	string(C)
strchr	function	string(C)
strcmp	function	string(C)
strcpy	function	string(C)
String	define	types
StringHandle	type	types
StringPtr	type	types
StringToNum	function	packages
StringWidth	function	quickdraw
strlen	function	string(C)
strncat	function	string(C)
strncmp	function	string(C)
strncpy	function	string(C)
strrchr	function	string(C)
StuffHex	function	quickdraw
Style	type	types
SubPt	function	quickdraw
SwapFont	function	fonts
swMode	define	sound
swOverrunErr	define	serial
SWSynthPtr	type	sound
SWSynthRec	type	sound
symbol	define	fonts
SysBeep	function	osutils
SysError	function	error
SysParmType	type	osutils
sysPatListID	define	toolutils
SysPPtr	type	osutils
SystemClick	function	desk
SystemEdit	function	desk
SystemEvent	function	desk
systemFont	define	fonts
SystemMenu	function	desk
SystemTask	function	desk
SystemZone	function	memory
taliesin	define	fonts
tan	function	sane
tan	function	trig(C)
tanh	function	tanh(C)
TEActivate	function	textedit
TECalText	function	textedit
TEClick	function	textedit
TECopy	function	textedit
TECut	function	textedit
TEDeactivate	function	textedit
TEDelete	function	textedit
TEDispose	function	textedit
TEFromScrap	function	textedit
TEGetScrapLen	function	textedit
TEGetText	function	textedit
TEHandle	type	textedit
TEIdle	function	textedit
TEInit	function	textedit
TEInsert	function	textedit

teJustCenter	define	textedit
teJustLeft	define	textedit
teJustRight	define	textedit
TEKey	function	textedit
TENew	function	textedit
TEPaste	function	textedit
TEPtr	type	textedit
TERec	type	textedit
TEScrapHandle	function	textedit
TEScroll	function	textedit
TESetJust	function	textedit
TESetScrapLen	function	textedit
TESetSelect	function	textedit
TESetText	function	textedit
testCntl	define	controls
TestControl	function	controls
testexception	function	sane
testhalt	function	sane
TEToScrap	function	textedit
TEUpdate	function	textedit
TextBox	function	textedit
TextFace	function	quickdraw
TextFont	function	quickdraw
textMenuProc	define	menus
TextMode	function	quickdraw
TextSize	function	quickdraw
TextWidth	function	quickdraw
TFeed	type	printing
THPrint	type	printing
thumbCntl	define	controls
THz	type	memory
TickCount	function	events
times	define	fonts
tk0BadErr	define	disks
tmfoErr	define	files
toascii	function	conv(C)
tolower	function	conv(C)
_tolower	macro	conv(C)
Tone	type	sound
TONEAREST	define	sane
Tones	type	sound
TopMem	function	memory
toronto	define	fonts
toupper	function	conv(C)
_toupper	macro	conv(C)
TOWARDZERO	define	sane
TPPrint	type	printing
TPPrPort	type	printing
TPRect	type	printing
TPrInfo	type	printing
TPrint	type	printing
TPrJob	type	printing
TPrPort	type	printing
TPrStatus	type	printing
TPrStl	type	printing
TPrXInfo	type	printing
TrackControl	function	controls
TrackGoAway	function	windows
trFunc	define	packages
true	literal	types
TScan	type	printing
twoSideErr	define	disks
UNDERFLOW	define	sane

underline	define	quickdraw
ungetc	function	ungetc(C)
UnionRect	function	quickdraw
UnionRgn	function	quickdraw
UniqueID	function	resources
unitEmptyErr	define	devices
unlink	function	unlink(C)
UnloadScrap	function	scrap
UnloadSeg	function	segload
UnmountVol	function	files
UNORDERED	define	sane
UnpackBits	function	toolutils
updateEvt	define	events
updateMask	define	events
UpdateResFile	function	resources
UprString	function	osutils
UPWARD	define	sane
useAsync	define	osutils
useATalk	define	osutils
useFree	define	osutils
UseResFile	function	resources
userItem	define	dialogs
userKind	define	windows
useWFont	define	controls
ValidRect	function	windows
ValidRgn	function	windows
vAxisOnly	define	controls
vAxisOnly	define	windows
VBLTask	type	retrace
VCB	type	files
venice	define	fonts
verArabia	define	packages
verAustralia	define	packages
verBelgiumLux	define	packages
verBritain	define	packages
verCyprus	define	packages
verDenmark	define	packages
verFinland	define	packages
verFrance	define	packages
verFrCanada	define	packages
verFrSwiss	define	packages
verGermany	define	packages
verGreece	define	packages
verGrSwiss	define	packages
verIceland	define	packages
verIsrael	define	packages
verItaly	define	packages
verJapan	define	packages
verMalta	define	packages
verNetherlands	define	packages
verNorway	define	packages
verPortugal	define	packages
verSpain	define	packages
verSweden	define	packages
verTurkey	define	packages
verUS	define	packages
verYugoslavia	define	packages
VInstall	function	retrace
vLckdErr	define	files
volOffLinErr	define	files
volOnLinErr	define	files
VolumeParam	type	files
VRemove	function	retrace
vType	literal	osutils
vTypErr	define	retrace
WaitMouseUp	function	events
watchCursor	define	toolutils
Wave	type	sound
WavePtr	type	sound
wCalcRgns	define	windows
wDispose	define	windows
wDraw	define	windows
wDrawGIcon	define	windows
wGrow	define	windows
wHit	define	windows
whiteColor	define	quickdraw
wInContent	define	windows
WindowPeek	type	windows
WindowPtr	type	windows
WindowRecord	type	windows
wInDrag	define	windows
wInGoAway	define	windows
wInGrow	define	windows
wNew	define	windows
wNoHit	define	windows
wPrErr	define	disks
wPrErr	define	files
write	function	write(C)
WriteParam	function	osutils
WriteResource	function	resources
writErr	define	devices
wrPermErr	define	files
wrUnderrun	define	disks
xOffWasSent	define	serial
XorRgn	function	quickdraw
yellowBit	define	quickdraw
yellowColor	define	quickdraw
ymd	define	packages
ZERONUM	define	sane
ZeroScrap	function	scrap
Zone	type	memory