Differences Between Using C and Pascal on the Macintosh

While most Macintosh references are written with Pascal as the language of choice, this database provides templates and examples in C. This topic covers some of the differences between using C and Pascal as related to programming on the Macintosh.

Naming Conventions

C is case-sensitive in all aspects. Structures, fields, data types, constants, and function names must be used with capitalization exactly as shown in this Guide. For instance, some implementations of Pascal allow you to use "DocumentProc" instead of documentProc in the *wDefProcID* parameter of a call to **NewWindow**. C requires exact character-case usage.

In general, the Macintosh convention is to use an uppercase character at the beginning of all system function names, global variables, and system-defined structures and data types. Constants and field names begin with a lowercase letter (when composed of two or more words or fragments, each subsequent "word" begins with an uppercase letter).

Functions and Procedures

Pascal users note that in C, all routines are functions. What you think of as a "procedure" is simply a function that does not return anything (the C idiom is to call this a `void` function). In this Guide, when we refer to "functions', we are including "procedures'.

This Guide uses function prototypes that are different from those used in Pascal-biased references. The simple rules are:

type RtnName(*parm*);	means	**function** RtnName(*parm*) : *type* ;
	-or-	**procedure** RtnName(*parm*);

Rather than include the data types of each parameter within the parentheses, this Guide shows the parameters on subsequent lines of the prototype. In these parameter lines:

type object ;	means	*object* : *type* ;	(different idiom)
type ***** *object* ;	means	**VAR** *object* : *type* ;	(pass a pointer)
	or	*object* : *trans-32-bit type*	(implicit pointer) (see below)

Call Parameters

A major difference between C and Pascal is that Pascal compilers decide whether to pass the address of a data type or the data itself. For instance,

```
SetRect( theRect, 100,200, right, bottom );       {Pascal}

SetRect( &theRect, 100,200, right, bottom );      /* C */

Rect * rp;                                        /* C option */
rp = &theRect;
SetRect( rp, 100,200, right, bottom );
```

Pascal sees that *theRect* is an 8-byte structure so it passes the address of that structure to **SetRect**. With C, you must use the "&" to specify that you are passing the address (or, optionally, pass the value of a pointer-to-Rect data type, as in second example, above).

Pascal bases its pointer-vs-value decisions on the simple rule that if the data type is larger than 4 bytes, it passes the address (a Rect is an 8-byte structure, so pass its address; a Point is a 4-byte structure, so pass its value). All Pascal VAR parameters are passed implicitly as addresses.

Another difference between C and Pascal is the order in which parameters are passed and the mechanism by which values are returned. These differences are hidden in both languages. This will have an effect when you write a "**filter**" or "**callback**" routine; i.e., a routine that will be called by the Macintosh System directly . C has a way to override its normal conventions and use the native (Pascal) conventions in these cases. Just use the **pascal** keyword when you create your call-back routine:

```
pascal Boolean myDlgFilter( theDlg, theEvent, itemHit)
DialogPtr theDlg;
EventRecord *theEvent;
short *itemHit;
{
        . . .
}
```

See **ModalDialog**, **TrackControl**, and other calls that use call-back routines for examples. See Custom Controls for another case where the `pascal` keyword is used. The Boolean topic has related information.

Strings

A standard C string is an array of characters, ending in a NULL byte (0) and C library functions such as strcpy(), strcat(), strlen(), etc., assume this convention. This type of array is sometimes called an ASCIIZ string.

On the other hand, the Macintosh assumes the Pascal convention of starting the character array with a "length byte" (and having no terminator). All Macintosh calls that use a "string" assume this Pascal convention. Therefore, your compiler may provide functions, such as PtoCstr() and CtoPstr() to convert between the two formats. In this Guide, you will often see string literals defined as:

```
char theString[] = "\pThis is a Pascal-style string";
```

C compilers will convert **\p** into a length byte, so the resulting array may be used in calls to the Macintosh Toolbox. See Str255 for related details.

Structures, Pointers, and Handles

Without going into great detail as to the idioms used in pointers and structures, here are some samples that will let you compare and contrast C and Pascal structure definition and usage:

Examples in Pascal

Examples in C

```
type
  PolyHandle := ^PolyPtr;
  PolyPtr    := ^Polygon;
Polygon := record                typedef struct Polygon {
  polySize   : INTEGER;             short  polySize;
  polyBBox   : Rect;                Rect   polyBBox;
  polyPoints : array [0..0] of Point      Point  polyPoints[];
end;                             } Polygon, *PolyPtr, **PolyHandle;

VAR hPoly   : PolyHandle;        PolyHandle hPoly;
    ptrPoly : PolyPtr;           PolyPtr    ptrPoly;
    theRect : Rect;              Rect       theRect;
    thePt   : Point              Point      thePt;
    x       : INTEGER;           short      x;

hPoly := OpenPoly;               hPoly = OpenPoly();
  .                                .
  :                                :
ClosePoly;                       ClosePoly();

ptrPoly := ^hPoly;               ptrPoly = *hPoly;

x := hPoly^^.polySize;           x = (**hPoly).polySize; /* or... */
                                 x = (*hPoly)->polySize;

x := ptrPoly^.polySize;          x = ptrPoly->polySize; /* or... */
                                 x = (*hPoly).polySize;

thePt := hPoly^^.polyBBox.topleft;     thePt.v =
(**hPoly).polyBBox.top;
                                 thePt.h = (**hPoly).polyBBox.left;

hPoly^^.polyBBox.top := 10;      (*hPoly)->polyBBox.top = 10;
ptrPoly^.polyBBox.top := 10;     ptrPoly->polyBBox.top = 10;
```

File Manager Structures

Many Macintosh references avoid discussing low-level File Manager calls altogether (see **File Mgr PBxxx**). And of course, Inside Macintosh uses Pascal in its examples and prototypes.

The Pascal "variant" data structures used in the File Manager calls have no direct parallel in C (structure "unions" are a bit more unwieldy). This can be a headache for C programmers, since you will need to choose carefully which structure to pass to a system function and you may need to use a different structure to interpret the results.