T E C H N O T E :
Where Has My qd Gone? And How Do I Use qd and QDGlobals Correctly?

By Merwyn Welcome
MERWYN@applelink.apple.com
Apple Developer Technical Support (DTS)

This Note addresses the genesis of and changes to the qd variable and the QDGlobals data type. The qd variable contains all QuickDraw global variables. This Note also demonstrates the correct usage of the qd variable and the QDGlobals data type.

This Note is of general interest to developers involved in Macintosh programming in C and C++.

Defining qd Now: Why and How?

At one time in the evolution of the Macintosh it was not necessary to define the variable qd. In fact, there was no system global variable called qd. As development environments changed from Pascal to C and internal data structures matured, however, a set of global variables were grouped together and called qd. The definition of this variable was, for the most part, hidden from the Macintosh programmer. Today, one by-product of using shared libraries is that you must globally define qd.

The Early Pascal Declaration

In the early days (before `qd`), a group of global variables were declared thus:

```
thePort: GrafPtr;
white: Pattern;
black: Pattern;
gray: Pattern;
ltGray: Pattern;
dkGray: Pattern;
arrow: Cursor;
screenBits: BitMap;
randSeed: LONGINT;
```

and could be accessed directly as follows:

```
InitGraf(@thePort);
```

But that was back in the days when Pascal was the programming language of choice for Macintosh development.

The C Declaration

When Apple and most Macintosh developers moved to C, it was decided that this group of variables should be tied together as a single data structure. The data structure was given the name `qd`, and declared thus:

```
extern struct  {
    char privates[76];
    long randSeed;
    BitMap screenBits;
    Cursor arrow;
    Pattern dkGray;
    Pattern ltGray;
    Pattern gray;
```

```
    Pattern black;
    Pattern white;
    GrafPtr thePort;
} qd;
```

without a data type in `Quickdraw.h`. It was defined for Macintosh programmers in the library `Runtime.o` (i.e., the actual storage for `qd` was provided by `Runtime.o`). This meant that to access `qd` or any of its fields, you didn't have to define it. The fields of `qd` could no longer be accessed directly as they once were, but only through `qd` as follows:

```
InitGraf(&qd.thePort);
```

At a later point the data structure was changed, given a data type and declared thus:

```
struct QDGlobals {
    char            privates[76];
    long            randSeed;
    BitMap          screenBits;
    Cursor          arrow;
    Pattern         dkGray;
    Pattern         ltGray;
    Pattern         gray;
    Pattern         black;
    Pattern         white;
    GrafPtr         thePort;
};

typedef struct QDGlobals QDGlobals, *QDGlobalsPtr, **QDGlobalsHdl;

extern QDGlobals qd;
```

With the introduction of the PowerPC came the Code Fragment Manager, and with the Code Fragment Manager came shared libraries. If an application used five shared libraries and each linked with the old `Runtime.o` module, each library would have a separate copy of `qd`, and not the one passed by the application to `InitGraf`! This meant that it was possible to have more than one `qd` variable defined at a time. There were problems with this approach. For example, if you had multiple `qd`s, updating one would necessitate propagating those changes to all `qd`s, which would be time-consuming on one hand. On the

other hand, if that wasn't done, then these `qd`s could easily get out of sync with each other.

Moreover, it was not a logical reflection of the world. For example, the `qd` variable has always been used as an abstraction of the user's computer screen, of which there is logically only one. `qd` was thus removed from the statically linked runtime library in the PowerPC environment. This meant that to link for the PowerPC environment, your application had to take on the added responsibility of defining `qd`.

To avoid separate code bases for 68K and PowerPC Macintoshes, it was recommended that the following definition be added to each application that made use of the `qd` global:

```
#ifdef powerpc
QDGlobals qd;
#endif
```

This worked fine until CFM-68K came along. The shared libraries in the CFM-68K environment suffered the same problems with multiple definitions of `qd`. During the development of CFM-68K, it was initially recommended that the previous definition be changed to the following:

```
#if GENERATINGCFM
QDGlobals qd;
#endif
```

As development of CFM-68K continued, it became apparent that the runtime libraries needed to be reorganized to better reflect the realities of how they were used, and to reduce duplication of code between the Pascal libraries and the three versions of the C libraries. In the process of this reorganization, the library `Runtime.o` was replaced with `MacRuntime.o` and `IntEnv.o`.

In addition, when the libraries were reorganized, the default I/O operations were redone, modeled on the newer PowerPC versions. Previously, if an application wrote to `stdout` or `stderr`, the output was written to a dialog box. To put up this dialog, the libraries needed to be able to draw to the screen, and so needed a `qd` global. Since the libraries already had one, it seemed logical to provide it to user. In the new I/O model, if an application writes to `stdout` or `stderr`, a file by that name is created, and the output sent there. The libraries no longer need to draw to the screen, and no longer need a `qd` global for their own use.

Because of the problems exposed by shared libraries, the changes to the I/O model, and to reduce the amount of CFM-specific code, it was concluded that it was no longer appropriate for the C Language libraries to define the `qd` variable. With the release of MPW 3.4, therefore, each program that requires `qd` is now required to provide a single definition of `qd` regardless of the runtime environment:

```
QDGlobals qd;
```

Using `qd` in Environments Other Than MPW

Currently neither Metrowerks nor Symantec support CFM-68K, but are expected to in the near future. Both Metrowerks' and Symantec's runtime architectures for the PowerPC are the same as MPW's, but as of this writing both continue to use the old classic 68K runtime architecture. If you need to develop in multiple environments, such as MPW, Metrowerks or Symantec, you need to use a preprocessor conditional statement. For example:

```
#if GENERATINGCFM

    QDGlobals qd;// Required for all CFM environments

#else

    #ifndef SYMANTEC_C || SYMANTEC_CPLUS
    #define __MPW_ONLY__
    #endif

    #if defined (__SC__) && defined(__MPW_ONLY__)
    QDGlobals qd; // Required for SC in MPW compilations
    #endif

    #undef __MPW_ONLY__

#endif
```

Summary

With the release of the MPW 3.4 runtime libraries, when writing a standard Macintosh application that uses standard Macintosh graphics, such as a graph port, you must now provide a single definition of the `qd` variable in order to allocate storage for it in your application. This definition is usually done in the global space of the file that contains the `main()` function.

Further Reference

- ETO #19, MPW Release Notes, p.6-26, 7-10.
- *Inside Macintosh: Volume I* page I-162, I-165.
- *Inside Macintosh: Imaging With QuickDraw* page 2-36, 2-62.
- *Inside Macintosh: PowerPC System Software*, page 59.

Acknowledgments

Thanks to Russ Daniels, Scott Fraser, Don Johnson, Alan Lillich, Alex McKale, and Bob Wambaugh.