# Technotes #1 – the QuickDraw™ 3D Viewer

*Written by: Nick Thompson and the QuickDraw 3D team*
*Date: 2/12/96*

The QuickDraw 3D Viewer is a useful way to incorporate QuickDraw 3D into your application with minimal effort. If you have a 2D application (for example word processors, page layout applications, web browsers, image databases - the Viewer has been used for all of these and more) then you'll want to take a close look at the Viewer. The references section will give some background reading, but remember to look in this technote, since we'll add new stuff here as we find it, and remove stuff from here as we fix it. This tech note describes a number of gotchas that you may encounter when working with the Viewer, some of these are bugs that we intend to address, others are quirks of the way the Viewer works. Hopefully this information will save you some head scratching and/or debugging cycles.

## Check the Viewer is installed before calling any Viewer routines

There are two ways to do this, either you can call `Gestalt` with the selector `gestaltQD3DViewer`, or you can check a routine against the constant `kUnresolvedSymbolAddress`. For either method you should weak link against the "QuickDraw3DViewerLib" stub library supplied as part of the QuickDraw 3D SDK, otherwise your application will not launch if the Viewer library is not present. If you don't weak link, you will get the system defined message and your application will not get launched, which means neither of these methods for checking for the presence of the library will work:

*The application “SimpleViewerSample” could not be opened because “QD3DViewerLib” could not be found.*

For information on weak linking, please check the documentation that came with your development system. There are several reasons why you might not want to rely on the default system message. First of all, the Viewer functionality may be an optional part of your application. For example your application may be an image cataloging program that catalogs QuickDraw 3D metafiles, Macintosh® PICT files, QuickTime™ movies, etc. If the Viewer is not installed, you may still want your users to be able to catalog QuickTime movies and Macintosh pictures. Secondly it is nice to give a customized message telling the user what to do. If you weak link your application you can run to the point where you need to display your custom message, and then either quit your application, or continue with the Viewer funtionality provided in your code disabled.

The routine below uses Gestalt to check for the presence of the Viewer library.

```
Boolean IsQD3DViewerInstalled()
{
      OSErr       theErr;
      long  gesResponse;

      if ((theErr = Gestalt( gestaltQD3DViewer,&gesResponse ))!=noErr)
            return false;

      return (gesResponse && ( gesResponse << gestaltQD3DViewerAvailable
));
}
```

The code snippet below may also be used if you weak linked against the Viewer library. The example compares the address of one of the Viewer routines against the symbol `kUnresolvedSymbolAddress` this is defined in the header file <CodeFragments.h> so make sure that you include this header file if you use this method:

```
if( (long) Q3ViewerNew != kUnresolvedSymbolAddress ) {
      /* your routines that use the Viewer library */
}
```

## Call MaxApplZone() before calling any Viewer routines

The Viewer checks to see if there is at least 24k free in the application's heap before allocating a new Viewer. If there is not enough free memory, Viewer routines will fail: for example the routine `Q3ViewerNew` will return NULL if there is not sufficient memory in the application heap.

## Offsetting the Viewer pane within a Macintosh window is broken in versions prior to 1.0.3

This was a bug in QuickDraw 3D that was fixed in releases of the Viewer including 1.0.3 and later. Attempting to offset a Viewer from the origin of the window would result in the Viewer drawing with it's origin at the origin for the window, even though the controls were drawn in the appropriate place. If you are experiencing problems of this nature, you should try to obtain and upgrade to the latest version of the Viewer.

## Set the port before handling a Viewer event

This is a bug in all release 1.0.x versions of the Viewer. You need to set the port, as illustrated by the following snippet:

```
/*
** There is a bug in versions 1.0.4 and earlier
** of the Viewer, so the port has to be set and restored.
*/
GetPort( &savedPort ) ;
SetPort((GrafPtr)myWind ) ;
wasViewerEvent = Q3ViewerEvent( theViewer, &theEvent);
SetPort( savedPort ) ;
```

If you don't do this it will appear that the Viewer is not responding to mousedown events, in both the control strip at the bottom of the Viewer and also in the Viewer content region.

## Setting the renderer

There was a crashing bug in versions of QuickDraw 3D prior to 1.0.3 that prevented the renderer for a Viewer from being changed. The following code snippet illustrates how you can set the renderer for a Viewer object:

```
// get the view from the Viewer
TQ3ViewObject     myView = Q3ViewerGetView( theViewer );
TQ3Status         myStatus;
TQ3RendererObject myRenderer;

if( selectedRendererType == kUserSelectionWireframe)

      /* set the renderer to wireframe */
      myRenderer = Q3Renderer_NewFromType(
                                    kQ3RendererTypeWireFrame );

else

      /* set Viewer to use interactive renderer */
      myRenderer = Q3Renderer_NewFromType(
                                    kQ3RendererTypeInteractive );

/* the renderer just set will "stick"with the viewer selection */
myStatus = Q3View_SetRenderer(myView, myRenderer) ;
/* finally ditch the renderer reference created above */
Q3Object_Dispose( myRenderer ) ;
```

We've always stressed in the documentation that it is possible to get the view object associated with a Viewer object, and that once you have the view object you can access parts of the view, for example the camera, lighting, or renderer set up for the view. In the example above we get the view from the Viewer, and then get the renderer from the view. We replace the existing renderer for the Viewer with a new instance of a renderer object that we created. In this way we can select between the two renderers supplied with QuickDraw 3D, the interactive and wireframe renderers.

## Setting the port

With QuickDraw 3D 1.0.3 and later you can no longer pass in a reference to a GWorld when using the routine `Q3ViewerSetPort().` If you pass in a CGrafPtr it associated with a window. The reason for this is that we changed the basis of the Viewer from being based on a PixMap DrawContext to a Macintosh DrawContext to improve rendering performance with 3rd party hardware accelerator cards.

You may also pass in NULL for the CGrafPtr, which will allow you to create an empty Viewer that can be associated with a window later.

## QuickDraw (2D) clip

The Viewer honors the window clip, but not the QuickDraw clip. This will be fixed in an upcoming release.

## Other (less obvious) uses for the Viewer

*Recipe 1 - using the Viewer to create PICT data to use in clipboard support*

Your application is more inter-operable with other applciations if it exports a number of types to the clipboard via the scrap. In general you should write the preferred type (3DMF) first, and alternative types (PICT is a good alternative that is handled by a large number of applications) later. You can use the Viewer to create an offscreen window, associate some 3DMF data with the Viewer, and use the API routine, Q3ViewerGetPicture to obtain a picture. If you need a picture fast, and you have a regular 3D application that is not using a pixMap DrawContext, then this may be your fastest way to get a PICT.

## Further Reading

*"QuickDraw 3D: A New Dimension for Macintosh Graphics" in Develop Issue 22,* by Nick Thompson and Pablo Fernicola. This gives and introduction to using the viewer, and a basic introduction to QuickDraw 3D.

*"3D Graphics Programming with QuickDraw 3D", Addison Wesley 1995,* by Apple Computer (lead author Tim Monroe). This is the definitive reference to the viewer.