Technical Note TN1156

Scribbling Into AWT Components

CONTENTS

This Technote describes how to draw into an AWT Component by means other than the Java AWT Graphics API. In particular, by discovering the QuickDraw `GrafPort`, origin, and clipping Region corresponding to the Component's visible area, you can use any means at your disposal (most likely QuickDraw) to draw things inside the Component.

Updated: [Mar 1 1999]

Introduction to Impure Drawing

Most Java applications are content (if sometimes grudgingly so) to use normal "100% Pure Java" APIs to draw their user interface. They use a combination of existing AWT control components like Button or Checkbox, and custom Component subclasses that use the graphics primitives provided by the AWT Graphics class to draw themselves. (They may also use components provided by higher-level class libraries such as Swing or IFC, which in turn use Graphics.)

However, some Java code needs to use drawing services provided by the platform's toolbox. Such code will usually be a Java library whose goal is to provide a Java API for features provided by the platform's toolbox -- Apple's QuickTime For Java is one such example; another would be an OpenGL interface for Java.

Such code will need to use either native methods or `JDirect` to make calls to the toolbox. Both of these mechanisms are documented elsewhere. When it comes time to draw, though, the problem appears: How does the drawing code acquire the resources it needs to draw into the Component? For instance, before making QuickDraw calls, you'd need to know the right `GrafPort` to use, and set up the `GrafPort`'s origin and clipping.

This is an issue that applies to any platform, not just the Mac OS (although the details of the resources necessary are of course platform-specific.) Sun Microsystems, therefore, defined an API collectively referred to as **DrawingSurface** that can be used to map from a Component object to native-window system resources.

Back to top

How To Do It

First, use the right kind of Component as your drawing surface. You need a Component with a native peer, so lightweight Components won't work. And doing your own drawing into components that AWT itself already draws into -- like Button or Choice -- is discouraged. The Component types that work best, then, are Canvas, Panel and any of the Window types (Window, Frame, Dialog.)

Getting to the DrawingSurface

The interface `sun.awt.DrawingSurface` is used to access native drawing surface information. This interface is implemented by non-lightweight component peers and by Images created for offscreen graphics. The interface contains only

a single method, `getInfo`, which returns a `DrawingSurfaceInfo` object. This is the main object you'll need to use.

Here's a Java snippet that shows how to get the `DrawingSurfaceInfo` for the Component `theComponent`:

```
import sun.awt.*;

...

DrawingSurface ds = (DrawingSurface)theComponent.getPeer();
DrawingSurfaceInfo dsi = ds.getDrawingSurfaceInfo();
```

(You can do the same thing from native code using JNI; it's just more awkward.)

How to Draw

You need to call the `DrawingSurfaceInfo`'s `lock` method before you start drawing, and its `unlock` method after you stop drawing. The `lock` method sets up QuickDraw's state and ensures the `GrafPort` is in decent shape for you to draw into it.

To prevent AWT code running on other threads from messing with the port (or other global Toolbox state) while you're drawing, you need to synchronize against the *Toolbox lock* before you call the `DrawingSurfaceInfo`'s `lock` method. This is described in detail in Technote 1153, *Thread-Safe Toolbox Access From MRJ* . The next snippet below shows how to do this.

IMPORTANT:
Synchronizing against the Toolbox lock acquires a central AWT semaphore which prevents other Java threads, as well as the native host application, from accessing the Toolbox until your `synchronized` block exits. This means that

- You should lock for as short a time as possible -- get in, do your drawing, then get out.
- You should not make any other AWT calls while the drawing surface is locked (nor should the thread doing the drawing do anything that could block against other threads of yours that need to make AWT calls, or you could deadlock).
- You should make absolutely sure that you unlock on the way out, by putting the `unlock` call in a finally clause.

Now that you've locked the drawing surface, QuickDraw is all set to draw into the Component. Its `GrafPort` is the current port, the local coordinate (0,0) is at the top left of the Component, and the `clipRgn` is set to the visible region of the Component (which will not include the regions occupied by any non-lightweight child Components.)

You can get the Component's bounding box in local coordinate by calling the method `DrawingSurfaceInfo.getBounds`. It's safe to make this call while the `DrawingSurface` is locked.

When you're done drawing, you don't need to restore the previous `GrafPort`. However, you must restore any state you changed in the Component's `GrafPort` (clipping, colors, etc.) And of course you need to unlock the drawing surface.

It looks like this, continuing the above snippet:

```
import QuickdrawFunctions;   // from JDirect Sample Code in SDK
import com.apple.mrj.macos.toolbox.Toolbox;

...
synchronized( Toolbox.LOCK ) {
    dsi.lock();
    try{
        Rectangle bounds = dsi.getBounds();
        QuickdrawFunctions.MoveTo(bounds.left, bounds.top);
        QuickdrawFunctions.LineTo(bounds.width-1, bounds.height-1);
    }finally{
        dsi.unlock();
    }
}
```

Accessing the WindowPtr

In some circumstances (e.g., if messing with the Palette Manager) you may need to find the Mac OS `WindowPtr` of the window the Component is in. This is *not* the same as the Component's `GrafPort`; MRJ 2.1 creates its own `GrafPorts` and never uses the `WindowPtr` directly for drawing. If you do access the `WindowPtr`, you should not use it for drawing -- use the `DrawingSurface`'s `GrafPort` instead.

> **IMPORTANT:**
> The `GetWindow` method was added in MRJ 2.1; it is not implemented in MRJ 2.0 and thus MRJ 2.0 will throw an error when trying to call this method.

You get the `WindowPtr` by doing the following. Note that you can also get the `GrafPort` and `GDevice` in the same way:

```
MacDrawingSurface mds = (MacDrawingSurface) dsi.getSurface();
int windowPtr = mds.getWindow();  // WindowPtr cast to int
int grafPtr = mds.getPort();      // CGrafPtr cast to int
int gdevice = mds.getDevice();    // GDHandle cast to int
```

Don't hang onto these values for too long -- they will not be valid after the Component's peer has been disposed, that is, after the Component or a parent is hidden or the window closed. For safety's sake you should only get and access them while the `DrawingSurfaceInfo` is locked.

> **IMPORTANT:**
> There is a bug in MRJ 2.1 that makes it essential that you call `getWindow`, `getPort`, or `getDevice` at least once on any `DrawingSurface` you ever draw into, if you plan on mixing native and Java-based (using `java.awt.Graphics`) drawing in the same Component or Image. Until one of these methods is called, MRJ 2.1 doesn't realize that native drawing is taking place, and it won't fully synchronize the Java drawing with the native calls.

Back to top

Compatibility

The `DrawingSurface` API is implemented in MRJ 2.0 and later, although the implementation in MRJ 2.1 is more robust.

The method `MacDrawingSurface.getWindow` was added in MRJ 2.1; it is not implemented in MRJ 2.0, and thus in MRJ 2.0 the class-loader will throw an error when trying to load a class that calls this method.

Not all Java implementations on all platforms support `DrawingSurface` -- Sun explicitly points out that sun.* classes are not part of the supported Java API set. Sun's JDK 1.1 and later do support `DrawingSurface`. Naturally, the exact OS calls you'll need to make to draw are platform specific, and the MacDrawingSurface class will not exist. You'll need to get documentation from the vendor of any other Java implementation to find out how to use `DrawingSurfaces` with it.

Back to top

References

Technote 1153: *Thread-Safe Toolbox Access From MRJ.* Describes the "Toolbox lock" and how to use it. A must-read if you're going to use the Mac Toolbox for drawing.

Back to top

Downloadables

Acrobat version of this Note (K). Download

Back to top
