Technote 1193

How to structure your handleCheckUpdate callback

CONTENTS

A JManager host application should call `JMFrameUpdate` from its window-drawing code, and should also implement a `checkUpdate` callback that will call `JMFrameUpdate` if the window's update region is non-empty. (`CheckUpdate` is called by the AWT to fix up the display immediately if part of the window may have been invalidated by some AWT action.)

What's the problem?

Both of these routines typically do something like this:

```
BeginUpdate(window);
...
JMFrameUpdate(frame,window->visRgn);
...
EndUpdate(window);
```

This results in AWT being called while the window is in a funny state -- the `BeginUpdate` call has clipped its `visRgn` down to just the area needing updating, so any drawing calls can only draw in that area.

The problem comes in if another thread gets control before the JManager call completes. This might be any other thread, and it may perform other AWT operations like graphics-based drawing, moving components, or updating controls. Since whatever it draws will be clipped down to the window's `updateRgn`, some parts of the window may not be updated, which looks ugly.

Back to top

How do we fix it?

The only workable fix turns out to involve a slight change in the way the host app calls JManager -- and the purpose of this Technote is to tell you to make this change. It's simple to describe:

Never call JManager while within a `BeginUpdate...EndUpdate` *operation.*

To do this, you'll need to restructure your drawing and `checkUpdate` code to be something like the following, which was taken nearly verbatim from MRJShellLib, the library that runs JBound apps:

```
BeginUpdate(window);
...
/** Called in response to a JManager checkUpdate request */
void RequestedWindow::checkUpdate() {
        RgnHandle update = WindowPeek(itsWindow)->updateRgn;
        if (! EmptyRgn(update)) {                            /*1*/
                // Determine the local/global offset:
                SetPort(itsWindow);
                Point globalPos = {0,0};
                LocalToGlobal(&globalPos);

                // Compute the global rgn used for the Java Frame's content:
                Rect content = itsWindow->portRect;            /*2*/
                if( resizeable && !theGrowboxIntrudes )
                        content.bottom -= 15;
                OffsetRect(&content, globalPos.h,globalPos.v);
                RgnHandle contentUpdate = NewRgn();
                RectRgn(contentUpdate,&content);

                // Clip the update region against the Java content region:
                SectRgn(contentUpdate,update, contentUpdate);  /*3*/
                OffsetRgn(contentUpdate,-globalPos.h,-globalPos.v);
                                             // move back to local coords

                // Call JMFrameUpdate *without* calling BeginUpdate,
                // which mucks up the visRgn:
                if( !EmptyRgn(contentUpdate) ) {               /*4*/
                        ValidRgn(contentUpdate);                 /*5*/
                        JMFrameUpdate(frameRef, contentUpdate);  /*6*/
                }

                DisposeRgn(contentUpdate);
        }
}

/** Main method called to handle incoming updateEvents. */
void RequestedWindow::handleUpdate( ) {
        checkUpdate();   // first redraw Java content
        BeginUpdate(itsWindow);
        ... // draw native content here
        EndUpdate(itsWindow);
}
```

In essence the `checkUpdate` method does the following:

1. Checks whether the window's `updateRgn` is non-empty; otherwise it returns.
2. Computes the region of the window allocated to the Java Frame and converts it to global coordinates.
3. Intersects the region against the window's `updateRgn`.
4. If the resulting region is empty, it returns.
5. Otherwise it calls `ValidRgn` with that region, to remove it from the `updateRgn`.
6. Then finally it calls `JMFrameUpdate`, passing in the clipped region.

Note that in the `checkUpdate` method we've avoided using `Begin/EndUpdate` at all. The main `handleUpdate` method still needs to use them, to correctly clip the drawing of the native content, but it calls `checkUpdate` *first* , before calling `BeginUpdate`. (Alternatively, you might need to draw the Java content after the native content; if so, you'll have to save a copy of the `updateRgn` before calling

`BeginUpdate`, since otherwise it'll be lost.)

Back to top

Why This Hasn't Been a Problem Before

This never showed up before MRJ 2.2 because the main thread, which handles JManager calls, runs at the highest possible priority, and with the older thread scheduler it used to be impossible for any other thread to get control while this thread was active. But with MRJ 2.2's new "fair share" scheduler, other threads sometimes get time to run.

Back to top

Further References

- Programming with JManager for MRJ 2.1
- MRJ 2.1 SDK

Back to top

Downloadables

Acrobat version of this Note (12K)

Back to top

To contact us, please use the Contact Us page.
Updated: 02-February-2000

Technotes | Contents
Previous Technote