TECHNOTE:
The Problem with & (Simple) Fix to Purgeable WDEFs

By Matthew Xavier Mora and Troy Gaul
devsupport@apple.com
Developer Technical Support (DTS)

This Technote describes a problem with purgeable WDEFs and explains how to fix it simply by marking WDEF resources as non-purgeable.

This Note is directed at application developers who include WDEF resources in their applications as well as developers who create WDEF code resources.

Defining the Problem

Like other definition functions, WDEFs contain executable code that needs to be locked down in memory whenever it is executed. If your application is using a custom WDEF marked as purgeable, the Memory Manager may purge the WDEF resource in order to allocate additional memory in your application heap. If your application is the “current”application, i.e., the one that is executing but may not necessarily be “frontmost,”the Window Manager reloads the WDEF resource if it has been purged before attempting to call it. Everything works fine.

If your application is *not* the current application, and your WDEF gets purged, you’re going to have a problem.

The Problem Scenario

The following is a sequence of events that may cause the problem to occur:

The Window Manger calls your WDEF when it needs to redraw your windows even if you application is not current. It does this by calling your WDEF directly whenever your windows get erased. If the WDEF has been purged, the Window Manager will try to reload the WDEF.

Since your application is not the "current" application, your application's context has not been restored and your resource chain is not the current resource chain. As a consequence, the Window Manager attempts to load a WDEF from the wrong resource chain.

Since it can’t find the WDEF, it calls SysError(87).

Why This Hasn’t Been a Problem Before

There are three reasons why this hasn’t been a problem in the past.

(1) Most applications use WDEFs for floating windows. As defined by Apple’s Human Interface Guidelines (HIG), applications should hide their floating windows before being suspended. Since the windows are invisible, the Window Manager under most circumstances won’t try to reload your WDEF if it does get purged.

(2) This problem might not be triggered has to do with the fact that applications in the background don’t normally allocate a lot of memory. For example, even if the WDEF is marked as purgeable, the Memory Manager may not actually cause your WDEF to be purged.

(3) Most applications don’t run with very tight memory partitions, so even though a WDEF resource is marked purgeable, it doesn’t need to be purged.

The Simple Fix

The simple fix to the problem of your WDEFs not being able to be reloaded is to mark your WDEFs as non-purgeable. This prevents the Memory Manager from purging the WDEF, so it is always available even if your application is not frontmost.

Additional Notes & Comments

The following are some important items that you may need to consider when working through the problem of purgeable WDEFs:

▲ Other 'DEF types (CDEFs, MDEFs, etc.) won't exhibit this problem because they don't get called when the application's process globals are not current as in the case of WDEFs. Theses resources types can remain purgeable.

▲ WDEFs in the System file (or in an enabler file and probably in extensions such as Aaron) are can safely to be purgeable because they are always in the resource chain — no matter what application's process is set up.

▲ An application with a purgeable WDEF may have code in it that makes it non-purgeable after bringing it into memory — in which case, this problem won't occur .

▲ Third-party memory utilities may force purging to occur more often, thus exacerbating this problem.

▲ If there is data embedded in a WDEF and the WDEF gets purged and reloaded, the data will have reverted to its on-disk values. Many popular development tools embed global variables in code resources. (If your WDEF needs to call SetUpA4, you're using one of these development systems.) A WDEF whose variables suddenly change to unexpected values can of course cause any of the crashes normally associated with unexpected values.

▲ In general, applications should not detach WDEF resources from their resource map (see Technical Q & A TB - 575) . However, it's worth noting that detached WDEF resources which are also purgeable can never be reloaded from disk.

Summary

The problem with purgeable WDEFs can be fixed by simply marking their resources as non-purgeable.

Further Reference

- *Technical Q & A - TB 575*

Acknowledgments

I would like to thank Troy Gaul for his wisdom in this area and Vik Rubenfeld for tracking down the problem. Thanks also to Brian Bechtel, Nitin Ganatra, Pete Gontier, Tom Maremaa, Tim Swihart, and Tom Maughan.