

1) Operational Goals

Provide a way for freeing event handlers from the chain of command safely and cleanly. While allowing the user to cancel out of the operation.

2) Fundamental, "key", or cornerstone architectural requirements (POSTMORTEM)

- Need a list of active event handlers
- Need to avoid disposing of objects which have methods on the stack
- Need a mechanism for safely disposing of event handlers in a chain of command
- Need to avoid attempting to remove an event handler twice.
- Need to handle application clean up effectively
- DEventHandler's need to know if they are in the fDeadHandler list to prevent them from attempting to do a death sequence twice (which could do things like attempt to close files which are already gone, as is the case for document objects) and causing problems.

3) Model of the implementation fulfilling these key requirements (POSTMORTEM)

- 2 lists of event handlers are needed, fEventHandlers, fDeadHandlers to keep track of the objects in the events of quitting the application and closing an event handling chain of objects without crashing the stack while calling delete on the dead handler objects.
- The event handlers add them selves to fDeadHandlers list when needed.
- DEventHandler's automatically install themselves into the event handler list as needed.
- Checks are made as objects are placed into the fDeadHandler list, to make sure that that object isn't already in that list.
- The fDeadHandler list gets flushed on null events to avoid disposing of an object which may have a method active on the stack (boom!!!)
- CleanUp loops over the fEventHandlers telling them all to kill them selves, which then gets flushed.

4) Impact/scope of the implementation on the existing body of code (POSTMORTEM)

- DEventHandler::DEventHandler handles the notification of birth to the DApplications object
- DEventHandler::KillMeNext handles the notification of the death to the DApplication object.
- DApplication maintains the lists of live and dead handlers via the member fuctions, Dapplciation::InstalHandler() and DApplication::RemoveHandler().
- DApplication also dose a clean up of all live event handlers on a quit.

5) Coding notes (gotchas, warnings, process thoughts, items to revisited later...)

- sometimes the clean up method in DApplication attempts to dispose of a non-object (it is caught in the oopsDebug hook __noObject, This bug doesn't happen very often and I haven't traced down its origins, but I WILL...)
- I found that problem! (I told you that I would...). The problem was that my iterator class can get fubar ed if the linked list changes as it is iterating over the list. The DIterator::GetCurrentThenIncrement method has no way of knowing if the object the incement references has been clobbered by the functions use of the current item in the itteration, and hence sometimes problems happen. I have fixed the problems in the class library, but keep in mind the restrictions associated with the use of the itterator class.

6) Testing notes(bug types, what made a bug hard to fix, what could have been done to catch it sooner....)

7) Process notes (what process did you follow, could it be improved)