

Off-Line Documentation template: Event Handling mechanism

1) Operational Goals

Parse out the events from the event loop and send event messages to whatever objects need to respond.

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

- Parse events from event loop.
- Have 1st crack methods do any needed housekeeping.
- Send message on to an event handling chain of methods as needed.

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

- The DApplication object parses the events into a set of 1st crack event handling methods, which then send the appropriate handle event message to the fTarget event handler object. (and exception is made for update events, but don't worry about it)
- The fTarget gets maintained by some of the 1st crack methods. fTarget is the first in a NULL terminated linked list of DEventHandler objects.
- A member gPassItOn is maintained (but not used very much) to allow the possibility of dynamically shortening the Event Handling chain as needed.
- fTarget typically will point to a DWindow object which has a WindowPtr member which has "this" in that window's refCon. UpDate and Activate events use this refCon to maintain the fTarget (in the Activate Event case) or to send the UpDate events to the possibly NON-fTarget Windows (while setting the gPassItOn to FALSE).

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

- The presence of a fTarget object requires that a common base class is needed to define and maintain the event handler linked list. ("base" is used instead of root, because DinkClass uses a common root class for debugging and dialog aids) Objects of this class have "Handle<<your faverated event>>" methods which pass on the event message to their fNextHandler if fNextHandler != NULL and gPassItOn == TRUE. (The maintenance of this linked list is a feature in itself and not discussed here.)
- DApplication has an event handling method for each type of event which dose its thing and passes the message on the fTarget->Handle<<whatever>> method.
- DApplication::UpdateEvt is one notable exception to the typical flow of control. UpDate events come from the OS and are indented to signal a window (possibly not the front window) to refresh a portion of itself. Because of this that message very possibly may not be meant to go to fTarget. To do this use of the Window's refCon is made. Window maintenance is another feature and the rest of the UpdateEvt handling will be deferred to that features documentation.

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

The UpDateEvt handling process relies on the window having a DWindow reference in its refCon. This will not be the case for dialog boxes and non application windows. But

under System 7 the Window kind checks currently work OK. I suspect that this may fail (but can be fixed) when using Communication toolbox windows. I'll check that when the time comes.

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)

This feature was developed at the time I was defining the process model being used. Hence; I really haven't anything to add to this area.