Off-Line Documentation template: Communications tool Box demo

1) Operational Goals
To implement a multi-document (channel) synchronous telecommunications application using the CTB and DinkClass. Support CTB based file transfer, copy/paste from terminal windows and conform to CTB user interface guild lines.

2) Fundamental, "key", or cornerstone architectural requirements (POSTMORTEM)
• For each window a set of CTB handles for the FT, CM, and TM need to be maintained.
• Provide CTB callback procedures for each manager (handle).
• Provide menu interface into the configuration of the tools and there operation.
• Support the "private window" CTB events, i.e. when the FTM puts up a dialog then do the right thing.
• Support activate and task switching events correctly when CTB tools are active.
• Do periodic sampling of the active CM channels and maintain the flow of incoming data to the terminal and file transfer tools.
• Provide support for selection and copying of the data within the terminal window.
• Provide support for pasting information to the terminal window and to the connection tool

3) Model of the implementation fulfilling these key requirements (POSTMORTEM)
1) The DComApp needs to maintain a list of sets of CTB handles, one set for each window open.
2) The DComWind maintains the TMHandle, instanicates it an sets a reference to it from that set of CTB handles in that list of sets of handles. (to centralize things all the CTB handles are freed by the DComDoc::KillMeNext method)
3) The DComDoc holds the reference to a CTB handle set, (which is also in the DComApp list of CTB sets).
4) The DComDoc instanceates FT and CM handles in response to menu selections. When done it sets the appropriate reference to the respective handles in that CTB set of handles.
5) The DComDoc frees all three CTB handles within its KillMeNext method.
6) The DComApp creates the CTB Handle set and initializes each CTB handle in the set to NULL.
7) The DComApp override the event loop, some first crack event handlers and one handle event method to keep the CTB happy.

4) Impact/scope of the implementation on the existing body of code (POSTMORTEM)
• overrode the most functions yet in the DApplication subclass while still doing the normal other overrides
• All the CTB maintenance requirements WHICH ONLY NEED A CTB HANDLE where typically implemented in the DApplication subclass.
• All Call back procs where implemented as static member functions of the class which instanciates the respective CTB Handle. i.e. FTM call backs are in DComDoc and TM call backs are in DComWin.
• User events, menu selections, key clicks ect. are handled by the normal event handler chain, UNLESS the over-ridden CTB related first crackers intercept the event without passing it on to the event handling chain.

5) Coding notes (gotchas, warnings, process thoughts, items to revisited later...)
• Apple documentation (Inside CTB, and Basic Connectivity Set programmer notes) was not enough to get a program working. Missing where key tasks which must be done to make an application like this work. (the periodic sampling of the active CM channels for incoming data for instance)

• The same is true for the Little and Swihart book "Programming for System 7", talked about all the extra things needed to be done for a CTB application without pointing out how data is normally gotten FROM the CM (sending data to the CM was discussed).
• Thank God I had the Surfer Pascal based CTB demo Application form Develop #3 to reverse engineer these undocumented operating procedures one needs to follow in a CTB application.
• CTB is unforgiving, after a crash of the application typical one needs to reboot the computer to clear the communication ports for use again.
• Working with some TM tools can be tricky, I had a few do-nothing call back stubs which made the use of the VT102 tool not operate correctly (it would not draw the status bar). The only clue I had was the surfer code was initializing its TMHandles with more NULL procs, so I did too and that fixed it.
• Programming with the CTB brought out a bug WRT the DDocument and DWindow close/killMeNext scheme. There are times when the objects get called in a bad sequence resulting in freeing handles twice. I had to modify the DDocument and DWindow classes to fix the problem. I believe that this problem is the same as that occasional oopsDebug signal I was occasionally getting with my other demo applications.

6) Testing notes(bug types, what made a bug hard to fix, what could have been done to catch it sooner....)
• Started to use Discipline on my Quadra at home but it wont work. It works on the IIfx used on site, I don't know what's the problem, but I'm disappointed.

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