Off-Line Documentation template: Files, documents and I/o

1) Operational Goals
Provide a easily customized way of dealing with files (and is stationary aware) for applications built from dink class.

2) Fundamental, "key", or cornerstone architectural requirements (POSTMORTEM)
• Need isolate the actual read/write methods from the file selection UI code.
• Provide for application signatures and file types
• handle stationary pads correctly
• know when that data is dirty and needs to be saved
• know when to save and save as..
• Need document to clean up the file stuff upon closing
• Need to support stationary pads

3) Model of the implementation fulfilling these key requirements (POSTMORTEM)
• Document objects get a few members fNeedToSave, fRef, fSpec, to support the requirements
• fRef == 0 implies that on a save operation a Save As... will be done, if fReff==ZERO then the file is always closed.
• File type will be gotten from the application object and stored in members fFileType
• the Signature will be gotten from the application as needed
• Opening a file uses 2 member functions, the first provides the UI and obtains the data needed to open and read the file and opens the file, the second is a virtual member dose the actual read of the data and puts it where ever it needs to go.
• Closing a document works analogously.

4) Impact/scope of the implementation on the existing body of code (POSTMORTEM)
• DDocument gets a few members which support most of the requirements. It Gets a StandardFileReply record, to make things easier when using System 7 file stuff. It gets a short for fFileRef to hold the file ref number needed to the write to disk, and a flag indicating whether or not the data is dirty and a save is needed (fNeedToSave).
• The fFileRef member also serves as a flag to do a Save As or not, if it is ZERO then do the save as otherwise assume it is a valid file ref number. This scheme requires that fFileRef be set to zero when ever a file has been closed, and other places just for safely.
• The actual writing/reading of the data occur in ReadData/WriteData which get called from generic file I/0 interface functions.
• DDocument::OpenFile, provides the user interface to the files location.
• DDocument::ReadData, actualy reads in the data.
• DDocument::SaveFile, provides user interface (if needed), and dose a bit of file system house keeping.
• DDocument::WriteData, actually puts the data out to the disk.

5) Coding notes (gotchas, warnings, process thoughts, items to revisited later...)
• This code looks too confusing, the documentation needs to be improved and a more intuitive file architecture is needs to be developed.

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)