File : c:\wrmail\mailbox.fld
Messages : .

|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 84
From darrylr Fri Dec 7 15:04:00 1990
To: gregw tonyw
Cc: darrylr
Subject: ole design issues
Date: Fri Dec 07 15:03:01 PDT 1990

I want to summarize what I'm aware of regarding pending design issues for ole. We need to figure out which of these we will resolve for the v1 spec and what the design is.

1. Provision for link management. This is the philipl set of issues. Another need relative to this came out of discussions on my press tour. Someone asked how we'd handle mailing a document with links in it across a wide area mail net, if the other side had no connectivity to the linked files. I said the mail gateway would have to notice the document had links and then pump across copies of the linked-to files. The analyst then asked if we were providing object enumeration api's to make this possible, and I had to say no. However if we solve the philipl issues then this should become possible. It occurred to me that a way to do this is to establish a convention where all files containing objects will also contain an object at the end of the file whose contents is a directory of other objects in the file. The ole library could manage this. Given that it would exist in a standard place in every file and have a standard format, we'd be able to provide a standard api for getting at the info and updating it if necessary. Part of this would be to establish the convention that you never store pathname info as part of a link. Instead you'd use a cookie that indexes into the link directory object. This is necessary to let us patch links without invoking the host app. Note that if we use a "hidden object" approach for storing the object directory, then it becomes a convention that our ole library must be called before the app closes a file. Ie, there should be some kind of ole "close document" routine. (See item 6).

2. Network naming. It's critical that when we store link naming info in a file that it contain any needed network naming info, and that this info be network-independent (ie, work for lanman and novell nets, despite their different network pathnaming conventions). This means that you have to expand the pathname to a fully qualified pathname (ie, expand current dir) and then test the pathname to see if it's on a network drive. We should extract out of the fully qualified pathname the servername, volume name, and pathname as separate items (some of which may be null). This allows us to store the information in a network independant manner, such that it can be recombined at runtime into an appropriate network pathname, or used to connect any needed network drives. Note that each of these components should be stored without leading delimiters (ie, no \\ for servernames, since for example lanman and netware have different ververname syntax). Note that it also becomes possible to handle links to volume-named floppies with an approach like this.

3. Icons/presentations. See my earlier response to bill where I suggested that a way to handle icons/miniatures/etc would be

_TONYW

X 549544
CONFIDENTIAL

by expanding on the idea of a server providing alternative presentation formats for an object. I really think we should think this through and put it in the design, since it solves virtually all the important scenarios without needing a third-party agent to do the work, and it leverages the paste special dialog we already have. Tony, I discussed this in more detail with greg, maybe you and I should go over it too.

4. Single versus multiple mdi instances. We already discussed this at length. I'd be happy if we could come up with a simple extension to our design that in general makes it a server choice of whether to home in to a running instance versus firing up a new instance, and such that if some client "really wanted" to try making the choice there'd be a way for it to do so (though we wouldn't encourage this). Orrie's proposal, while it may not quite work, comes close to this, since it has the client try to do two initiates in sequence; so in a case where a given server is willing to accept multiple initiates, it's possible for a client to choose to take advantage of that (by using the special initiate) or in fact to get a separate instance (by using a regular initiate). So although we may say that one should always try one initiate and then the other, it's still possible for a client to bend the rules.

5. Format for embedded data. The spec doesn't say that data passed via the clipboard for embedding should be a virtual file image, however unless someone can point out a valid issue I think that we "should" say that. This is necessary to let a third party like the file manager make objects on behalf of apps, without invoking the app.

6. OLE api/library robustness. I'm concerned that our ole api and library adopts a particular set of implementation choices that many mainstream apps may find unacceptable and therefore they'll end up writing to the protocol. Ie, the api doesn't provide any options on how to handle things where there's more than one choice, and the choice we've made in the interest of simplicity may be the wrong one for many cases. I don't have a list of what these choices should be, but odds are we can learn that from isv's who look at the library and tell us why they can't use it. My point is that we "must" get this feedback because we must get isv's to like the library well enough to use it in most cases. We may also be missing some functions that we need to make the library robust and expandable in the future. For example, shouldn't we have entry points that get called when you open and close a document? This would give us the chance to do any document-specific init and termination processing (like reading and saving a directory object, for example). Do we also need a library-level init and term pair of routines (os/2 lets you have these but does win? Ie, do we need to make these explicit parts of the api?)

C:\TMP\DB005452. Thu Jan 10 15:11:00 1991