Erik Stevenson (LCA)

From:	Michael Mathieu
Sent:	Saturday, January 25, 1997 4:51 PM
To:	Ralf Harteneck; Brendan Busch; Peter Pathe; Andrew Kwatinetz; Antoine Leblond; Steven Sinofsky; Jon DeVaan; Duane Campbell; Chris Peters; Craig Unger; Richard McAniff; Daniel Bien; Eric Michelman; Brian MacDonald (Xenix); Bill Bliss (Exchange), Nathan Myhrvold; Bill Gates; Jon Reingold; Dean Hachamovitch; Brad Silverberg; Manish Vij; John Ludwig; Mark Walker (Word); Paul Maritz
Subject:	Thoughts from the Word 9 offsite -- applying it to all of Office 9

On Thursday I spent all day at the Word 9 pm offsite. I think we made some pretty big breakthroughs there (at least in the ways that I've been thinking about Word and FrontPage.) I think we came up with some important things that also impact the way the other apps might think about their plans for Office 9.

I'm not sure where everyone is in their thinking right now, so I'll just put the basic flow of thinking down below, and just let me know if you don't buy into various pieces or need more explanation, etc. We went through something like the following discussion:

1 - HTML is important to our apps business (we talked about this just to make sure everyone was really bought in. We are.)
2 - To be a player, you have to write HTML natively (this is *playing* vs. *dabbling* that we do today)
3 - There's only one HTML, and it's defined by the browser (i.e. no inventing your own tags)
4 - Since it's HTML, it's the browser's responsibility to view the documents (A **key insight** from AndrewK)
5 - Since people print what they view, it's the borwser's responsibility to print the documents
6 - How do you get all of our existing features into the browser then? No consensus here, but maybe you don't get them all into the browser. Possible alternatives: a) revert to Office97 formats if you hit a feature you can't render; b) don't support all the features (this would probably require use to make a new product, for marketing reasons, even if it were from the same code base (this would be more like "WebOffice" than the "New Internet Application", I think); c) add the features to the browser (three options there -- get Trident team to do it; Office devs party on the Trident codebase to add features; figure out an architecture that lets us install low level extensions to the browser to give us what we need (no one knows how to do this today on either side of the equation.))

4+5 imply important things like:

- Apps need a way to preserve editing information within valid HTML (FP WebBot trick w/comments)
- File Open/d-click from shell of Office apps works just like any other HTML page -- it comes up in the browser
- If you want to edit the doc then open in the browser, and then hit Edit button (viewing outnumbers editing)
- Might have smarts to edit right away if we see that you're the author of the doc
- Need a new meta tag to indicate the "preferred editor" even though it's just an HTM file (e.g. don't want PPT files being editing in FrontPage, b/c we won't understand all of your WebBot junk, and won't have the ideal UI for editing it. But technically it would be possible.)
- All printing smarts from today's apps should migrate to the browser. e.g. footnotes would be displayed in one way on screen, but browser should be smart enough to print them at the bottom of the page. Same goes for all the PPT color printing controls and smarts about how to divide things up into speaker notes, slides per page, etc. Users should get all of this in the browser. We'd need to figure out a new architecture for doing this -- and yes, it probably wouldn't be as good in the beginning as it is in the standalone apps, but we'll fix that with a few turns of the crank.
- Viewing and Printing are just two more examples of things which get "horizontalized" (to quote Andy Grove), when the file format for apps get horizontalized (Manish's insight)
- [Just came up with this one while I was writing this --] Of course this all has a big impact on our programmability story. The object model for all of our runtime capabilities should be aggregated onto

the *browser* object model. The portion of the object model that affects the editing environment is really entirely separate and distinct. That's not to say that editing isn't part of the browser object model. It's the editing *environment* that keeps it's own OM outside of the browser. So, the runtime object model is yet another thing that gets horizontalized by the browser with the common HTML format. Of course, the individual apps could provide runtime OM's which have redundant functionality to the built in Trident OM, but that makes sense b/c it's much easier to code with task-specific OM's, rather than just very low level control.

What does all of this mean for PowerPoint?
- Save natively as HTML (don't know if this is already in your plans)
- File Open goes to the browser. Edit goes to PowerPoint
- Transitions are built into Trident. You'd be less feature rich with Netscape, but we ship IE with our products.
- Need to figure out how to get required viewing and printing functionality into the browser. Short term hack way might be via Java applets, but you're more likely to want to put this into our browser -- that really improves our platform story, b/c now it becomes a great platform for people to target with presentation graphics packages.

What does all of this mean for Excel?
- Save natively as HTML
- File Open goes to browser. Edit goes to Excel
- Excel's special printing knowledge needs to work somehow in the browser. Will browsers ever handle the 2-D scrolling region as well as Excel? That might be a particular investment area we want to look at for the future. Or the 2-D-ness might just be particular to the online editing environment that Excel provides, rather than the viewing, which takes place in screen/page-size chunks.
- Need to decide level of functionality that goes into browser vs. addons. e.g. sorting, pivoting(?), filling in forms, etc. Today you can get Java applets that do a lot of what you want for basic list management. Data entry is the huge terrible part.

What does all of this mean for Access?
- DBC, Reports, and Forms saved as native HTML
- Table creation, query building, and programming are still native to Access (as is the MDB format for tables, indices, etc.)
- File Open goes to browser, as does everything outside of Design mode
- Need to build in some intelligence into the browser for how to handle large data sets, and how to do all of the data access remotely; also all of the banded report printing -- probably very different for printing reports than how they'd ideally show up in the browser (e.g. report header in a frame at the top. Same for page header (what does that mean for a bottomless report in a browser?) and all the footer stuff.)

What does all of this mean for Outlook?
- HTML as native format. My understanding is that they're already going fullbore on this.
- File Open goes through Browser -- that's not just for email and news, but views and view elements as well.
- Move to a web UI for viewing. Seamless integration for editing tools. This is in contrast to today's model where Outlook is almost like a wrapper of its own (besides the browser) and has it's own non-standard views. This is longer term, but basically all of the great views features in Outlook should move *into* the browser, rather than the browser becoming just another view in Outlook. Despite that it's what Netscape is doing, it's not the right thing to have this inside-out model. The browser is the one thing that should control views, and provide the runtime functionality for interacting with them (e.g. forms.)
- More file printing intelligence moves to the browser.

What does all of this mean for FrontPage?
- We're already HTML native format
- File Open through the browser is a model that we've been using for one area of FP3 improvements. You end up browsing around and then hitting the Edit button. We're making changes here in FP3 so that you can edit the page directly without having to first open the web in the FP Explorer. That makes things way faster.

- Printing -- well our only printing is through MFC. It's got tons of bugs, but we don't really get complaints. IE already does a better job than FP at printing pages. It just makes sense to put it in IE.
- We do need to think about hosting our Explorer views in the browser somehow. This is similar to the Outlook case where the views really belong *inside* the browser, rather than outside.

In a sense, this whole email can be summarized as "What changes when you don't have your own file format?" That's a consequence of HTML. AndrewK's insight that the browser should do all viewing is really crucial, and I don't think it's something that we've ever thought about before (witness our Word, Excel, and PPT Viewers.) It makes so much sense. And thinking through the implications of that for our apps will make us all work better in a world where don't have our own file format. This also lets us think a lot more about how the browser could become a platform for "real" applications, with a whole range of sophisticated needs that wouldn't necessarily be built into the browser. This is just a starting point.

Thanks,
-Mike