TO Chris Jones, Joe Belfiore
FROM Tandy Trower
SUBJECT Review of IE 4 UI
DATE 4/25/97
CC Bill Gates, Paul Maritz, Brad Silverberg, David Cole, Aaron Contorer

Per your request here is my review of the IE 4.0 UI. In preparation for this report Joe provided me with a demo of its interface and I also reviewed recent data from the IE usability team.

Summary of Review

Overall, IE 4 contains some good improvements. Features like auto-completion, Back button history pop-up menu, and dragging and dropping of links to the IE toolbar are positive innovations. They clearly enhance a user's ability to navigate the web more efficiently. Similarly, the integration of dynamic HTML (aka Trident) provides a richer model for creating highly interactive web pages. However, other features included in IE 4 add complexity to our overall interface. For example, the current implementations for Smart Favorites and Subscriptions as well as Desktop Components have serious usability problems. Further, the desktop-web integration story is weak. Integration is supported in some areas, not in others. The present form of integration also brings with it inconsistency and complexity. It needs some serious work before it will be compelling or simple enough.

I see no reason to force the proposed dramatic UI changes to the desktop, particularly when so much more work is needed. As a result, I strongly recommend that you consider redefining IE 4, focusing only on the browser's enhancements and its associated applications and deferring the web integration portion until Memphis. This will allow the IE team to do a better job on both aspects and provide a better rationale for the relationship between IE and Memphis.

The following information includes more detailed comments on elements of the IE 4 UI.

Browser Improvements

Auto-Complete

Auto-complete is a very nice feature, especially for experienced users. However, you should reconsider using Down Arrow to page through completion alternatives. The Down Arrow key is supposed to display the drop-down portion of a combo box (though it is inconsistently implemented throughout Windows). With auto-complete, it only drops the list when there is no matching completion in the list. This might sound reasonable, but it will take users significant time to learn the subtlety of its interaction and because it is inconsistent with drop-down control behavior elsewhere. For new functionality like this, I recommend avoiding redefining the basic interaction for drop-down controls. Instead define new interaction. For example, consider using Shift+Down since this key is not defined for the control and since the key combination maps better conceptually. For example, in other contexts Shift+Down performs an extended selection. In the auto-complete context, proposed completions also appear as selected text.

Back Button History Pop-up Menu

The Back button history pop-up menu is also an excellent enhancement. However, you should consider implementing this using the button-menu combination found in the Office97 applications.

An Office Button-Menu Combination

This makes the functionality more discoverable and the interaction more obvious.

Drag and Drop of Links

The ability to drag a link to the IE toolbar is a good improvement. However, it still needs some work. The right drag interface does not follow the standard convention of displaying a menu when you drop. Instead it has the inconsistent behavior of posting a message box requesting confirmation. This is fine for the left drag and drop, but inappropriate for right drag that already gives the user the option of confirming or canceling the drag operation. Finally, it seems unnecessary to refer to this functionality as a "Quick Link." Don't we already have enough terms for a link? Also why does this message box indicate Quick Link in its title bar and the IE icon in the window. Why doesn't this message conform to our normal conventions for message boxes?

Dynamic HTML

Dynamic HTML is also an excellent addition for IE. However, in the examples I looked at I found a serious interface issue having to do with how clickable text is presented. For example, moving the mouse over text does not change to the pointing finger pointer, put a frame around the text, underline the text, or any other form of feedback for conveying that the text was clickable. Instead the pointer remained as the text pointer implying the text was just static information. So the page had to include instructions to tell me to click. It is bad enough that we have so many ways of representing a clickable item, but none of them appeared here. Even if this is left to the developer to support, there should be some default feedback.

Search

The Search Pane is a nice feature, but it still needs some work. For example, when the pane is displayed, you get an additional toolbar area, but it is non-functional. Either get rid of it or put some buttons on it. Otherwise, don't waste the user's view space. Similarly what seems strange is how you start a new search. I would expect a button in the Search pane's toolbar, but instead there is another frame within the Search pane that says "Click here to start a new search", but it doesn't even look like a button or a link. Finally, repositioning the Search frame is very difficult. I dragged and got it to flip to the top of the viewing frame, but had great difficulty trying to dock it back to the side.

Drag and Drop in Menus

This seems like a fundamentally bad idea. Here we take a very simple interface that works pretty well and make it harder to use. It is not surprising that users find it difficult to learn, as it requires careful mouse manipulation to accomplish the task. For most users, it means they have now lost the ability to button down in a menu and drag to make a selection.

However, it is not only the greater skill required to discriminate operations, this proposed change to our menu interface also makes menu interaction more unpredictable and unpredictability in an interface makes the interface less discoverable and more difficult to learn. First, for example, we provide no visual cue as to which menu items can be dragged and which cannot until you start dragging in the menu. Further the drag and drop support for repositioning menus is also asymmetrical, that is, only left drag works, not right drag. Such inconsistency also reduces the overall usability of right drag when it works under some conditions and not in others. Somehow the user must learn our invisible rules for this behavior.

We seem compelled to make our menus harder to use. We already tend to overload them and cascading menus (which are known to be harder for users to master) are creeping in more and more. There have been proposals for scrolling menus, drag off menus, and pop-up menu for menu items. We also reduce the effective readability by sticking icons in the menus (as well as reducing the effective meaning of what an icon represents). By adding complexity we are dooming the simplicity of this interface and dragging menu items DOES increase complexity.

Desktop Toolbars (aka Deskbars?)

I have long suggested that the taskbar should be considered a toolbar of the desktop, even to the point of having originally recommended that we call the "Taskbar" the "Desktop Toolbar". (It is interesting to note that in a recent usability study, one third of the users didn't even know what the taskbar was.) So, while I endorse the evolution of the taskbar to a toolbar, aspects of the current IE implementation add complexity

Users can now start applications from icons they drag to the taskbar. While this seems like a good idea, it begs the question of the purpose of the Programs entry on the Start menu. The value of this new feature points out the inherent weakness of the Start menu. So now we have overlapping functionality. Perhaps more thought should go into eliminating the Start menu or integrating it better into the toolbar design.

I also found it difficult to manipulate the sections of the new taskbar design. Once I separated them, I had difficulty reintegrating them. I found that I could reposition the taskbar only when I started dragging in certain areas of the taskbar (maybe a bug?). It was also difficult to determine whether I was docking a toolbar adjacent to the taskbar versus docking it into the taskbar. Sometimes in an attempt to dock with the taskbar, I ended up windowing the toolbar and then it disappeared because it was behind the taskbar, or because I had the Restore Desktop button pressed.

The idea of dragging a link to the edge of the screen to become a toolbar is interesting, but I wonder if it has sufficient value. Here again, I found inconsistency in that right button drag did not mirror left drag. Right drag should always parallel left drag, except to offer a menu instead of performing an operation. I found that right dragging a link to the edge of the screen did not offer me the choice to create a desktop toolbar, but only Create ShortCut(s) Here and Cancel.

Smart Favorites and Subscriptions

Confusion between these overlapping concepts is by far one of the most serious problems in the IE UI. The motivation for Smart Favorites and Subscriptions is great, but perhaps they need to be unified or clarified in terms of their relationship. In addition, the user needs to be made aware what these features will do, or we should not turn them on by default.

Another problem with Subscriptions is that it is difficult to get a good overview of the times for all the components you have scheduled for updating. While the Subscriptions folder lists the times, you don't see them relative to each other. A better approach would be to offer a time organized view of this folder.

Desktop Components

The idea of active HTML components embedded in the desktop is another good idea in concept, but very confusing in it present implementation. First, depending on where you right-click on one of these, you may get the pop-up menu for the page, an element of the page, or you might even get the pop-up for the desktop (a bug?). This can be particularly confusing for a user. For example, when you right-click and get the page's menu, its lists Back and Forward, but users could interpret these to apply to changing the visual z-order of components. To add further complexity, over one third of this menu's commands are disabled.

However, what's worse is that you can't get a pop-up menu for the component. At a minimum a user should be able to right click on the move and size drag handles the component displays (that annoying flash as you move your mouse across your desktop). The resulting menu should include commands that allows you to manipulate the component, such as Hide to turn off a component's display or Delete to remove the component. Perhaps it should also include a Desktop Properties command since it is non-obvious that this is related to component layout. Further that corresponding Desktop Properties page also needs improvement because it is unlikely users will understand the relationship between the URLs displayed in the property page and the associated component. Perhaps you could use the image of the display (already used in the property sheet) to visually indicate what component relates to what URL.

There should also be a command that allows you to open the component into a window. Attempting to resize the component, particularly if you navigate to a link in the HTML of the component is frustrating. In fact, it is questionable whether these components should do anything other than display information. Navigation can be confusing. Further, that components can have clickable items means that the desktop is no longer a safe place to click to cancel a selection. Let's say you selected five icons on the desktop and want to cancel your selection with the mouse. Currently you just need to click on a non-icon portion of the desktop, but with desktop components, you might not be able to know whether that click might navigate within a component.

Supporting direct interaction to desktop components seems prone to problems because it is difficult for users to understand that the desktop now has THREE layers, the desktop itself, the icon layer and the desktop component layer. Perhaps desktop components should behave more like embedded OLE objects with respect to their user interaction. In other words, they should not allow direct interaction, but require a click or double-click first to make them interactive. That would allow them to display their content, but avoid collisions between normal desktop interaction. Similarly, OLE objects support a particular command to activate them and only then does the object expose its own interfaces. Finally, in our OLE document model, clicking outside of the object, automatically returns it to its non-interactive state. A lot of design work went into our OLE compound document design and it was designed to resolve many of the same interaction problems in the desktop-desktop component implementation. Why aren't we following our existing document model for desktop components? (Note that non-interactive does not mean that components cannot change their visual appearance. It only means that a user doesn't get confused between interacting with the container and the embedded objects.)

Channels

This interface still needs some work. It seems so divorced from the normal desktop interaction that it weakens our story on shell-web integration. The integration here should be more seamless. The lack of integration here reminds me of how the Bob group attempted to create a separate environment on top of the Windows desktop.

Prior to Win95's release, I recommended the concept of multiple "project workspaces" (aka multiple desktops), visual work areas where the user can place documents (or other objects) that they use together such that opening these workspaces reopen the objects kept there. Let's say you are merging information from an Excel report and a Word document to another Word document and at the end the day you shut down your system. There is no easy way to restore what you were working on. While you can use the Documents entry of the Start menu to reload them or their respective MRUs, it takes more work to restore them then if I could just open one object. Think what a productivity improvement it would be if you could open the workspace where you stored these and they all came back just as you left them.

The desktop itself could be considered a workspace, so rather than a single work area, a user could have many. This concept isn't new and it isn't even original with me. Outlook actually comes close to implementing this, though its associated objects are hard-coded by the app rather than user selectable. I bring this topic up again because it not only demonstrates the question of whose responsibility it is to implement such a solution, but also because

Channels could really be considered a type of workspace. Unfortunately we've made the Channels interface into just another thing in the interface with its own rules and conventions. Rather than creating generalized solutions, we add complexity by implementing specific solutions that overlap with other parts of our interface. The net result is increased complexity in our interface.

Mail/Outlook Express

While it is nice that we provide a mail client, it is a significant problem that we provide a user with no help in understanding which client they should use. A typical user is likely to end up with clients from Exchange, Outlook, and IE Mail/Outlook Express. How are they possibly to know which to use and which they can remove (or even how to remove them)?

FrontPad

With our entire set HTML authoring products, the motivation for including this one isn't clear. Are we trying to reduce sales of Visual Studio or FrontPage? In any case, this application has a number of problems in its UI that need to be addressed. If they are not, this just becomes another bad example of a Windows application. We encourage developers to follow our UI guidelines, but we continue to include applications that don't meet them. For example, why does the application use the same icon for the application as for its documents? Why does this application use MDI? Its use here just makes this

application more complex for users. If you want more details, we should schedule some time to go through the application.

Desktop/Web Integration

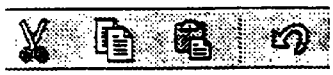
The concept of unifying the user's desktop and web experience sounds good and reasonable, but it's not clear that this is what users want and certainly is not what they expect. Many users expect to just get browser improvements with IE4 and I've heard many a remark from users that they don't want to view their folders to look like web pages. People are more likely to expect and accept changes to their core UI in Memphis than with a browser update.

If we really think that shell/web integration is a good a good idea, but we need to do a better job of communicating and demonstrating its value. For example, in the HTML views for folders, we display contextual information in the HTML pane of the folder based on the current selection. Yet when the user selects a link in the HTML pane it isn't always contextually related. For example, in the Control Panel, selecting an icon displays its description, yet if the user clicks on the Connect button, the resulting page is no longer contextual to the selection.

The integration story is also weakened by the inconsistencies it brings with it. For example, when viewing an HTML page in the browser, if you select View Options, you get the IE's property sheet. However, if you view a folder you get the system's folder viewing properties. Similarly, if you open a folder within the browser, it displays within the browser's window, but if you open an HTML page within a folder view, you get a separate window, even if you have set the option to reuse the window. Where's the integration here?

Another confusing aspect is how to turn off this web view. Some folders (but not all) include a menu command named "Web View" that I could uncheck, but in the Options dialog box for the same folder I also found an option button, called "Use Standard Windows View" in a group box labeled "Web View". This option was enabled even when I had turned Web View off. It seems unlikely that users will understand relationship between these options and returning to the conventional view of folders.

I also found some needless inconsistency in the integration. For example, the images for a folder's toolbar are different than found elsewhere in the interface. Contrast the following Clipboard command images.



IE 4 toolbar images

Office/Win95 toolbar images

Not only are the IE images different, they are also, larger and less well refined. While they have similarities in basic appearance, the aesthetics differences can make users wonder if there are differences in functionality. But there is a further difference. In IE, the images are monochrome until the user moves over them. In Office and Windows 95 toolbar images always have color. Even if you consider the IE method of tracking better, because it has not been consistently applied through Windows, it weakens the shell-web integration story.

Ownership of the Windows UI/Usability

The shell-web integration may be a good idea, but IE 4 fails to really accomplish this many of its interface changes are not carried through the entire Windows interface. I think the reason for this is that the responsibility for the usability of our interface is split between IE and Windows groups. Take the simple example of the current visual design in our interface. It's a mess. We have a hodgepodge of 3D buttons and flat fields. Sometimes controls change color when the mouse tracks over them, sometimes they get 3D frames. I read a proposal last year from a former IE graphic designer on how better to unify our visuals,

even between web pages and the Windows UI, but nothing ever happened to that because it would have required fundamental changes in Windows and that's the responsibility of another division

IE cannot address many issues of Windows usability. Many UI design issues straddle the IE and Windows groups. For example, improvements such as a better type definition other than reliance on 3 letter extensions or file system changes so that links don't break requires more than fits within the browser UI improvements. If we want to evolve and improve our Save dialog design. Who addresses this: the Windows group or the IE group? How can we have different groups trying to address the overall issue of the usability of our UI? For example, considering the benefits of auto-completion, which takes responsibility for making this functionality standard in for our drop-down controls?

Here's another example. The advent of the web (or you could say the advent of multimedia) has changed the key design objective from being based on the "printed" page (WYSIWYG) to the "viewed" page. As a part of this, we need to do more thinking about how to improve output on the screen. Several years ago, Chris Larson suggested the idea of anti-aliased fonts. It seems time to consider ideas like this. However, is this an IE design issue or a GDI issue or both? How do we prevent things like this from falling between IE and Windows groups?

It is understandable that it may have been necessary to separate the IE UI team in order to focus on the issue of making our browser competitive. However, it is time that we need to re-integrate the Windows and IE UI teams, otherwise our goal of shell/web integration becomes a shallow story and we end up with an interface that is eternally inconsistent with itself and redundant, overlapping alternatives for expressing the same concept.