From:	Christian Fortini
Sent:	Tuesday, August 26, 1997 3:56 PM
To:	Trident Leads; Adam Bosworth; Robert Welland; David Bangs; Chris Jones; David Cole; Hadi Partovi; Chris Guzak; Joe Peterson; Joe Belfiore; Bob Heddle; Steve Isaac; John Cordell
Cc:	Holly Marklyn
Subject:	Re: Trident directions for IE5

As we are slowly starting to see the light at the end of the IE4 tunel, I have been given some thoughts to future directions for Trident in IE5 and IE6. Here is a first stab at it.

This memo lists some of the design, architecture and feature directions that I can see Trident going into. It is meant to generate discussion and start the thinking process on specific areas. It is not comprenhensive by any means. Some sections at the bottom are incomplete or void. I am still working on them and will send an updated version of this document incorporting them and feedback later.

I have intentionally not mentioned any time line in here since I don't think we will be able to determine what we can achieve in IE5 vs. IE6 until we move forward on the planning and starting costing and prioritizing the task items involved.

Feedback, suggestions, comments, criticisms, etc., appreciated.

Thanks
Christian

Extensibility

Componentization

We learned a couple of lessons with Trident v1: 1/ accommodating 40 devs in a single code base is not a trivial exercise and 2/ responding to the very different needs of every different application or tool building on Trident Editing is next to impossible.

We have to stop adding non-browsing features into Trident and start taking some of the existing ones out. We should shrink the core Trident code base down to a very compact (and fast) HTML rendering and manipulation engine and hopefully limit the number of people in this code base.

To achieve this, we should expose a low level API (or object model) that allows external components to manipulate the HTML tree. We should also evolve our event model to let external component override any of our default behaviors. We should then extract our user-level editing support (things like the "bold" or the "apply bullet" commands) out of the core Trident into a new external component that is build upon these new services. The same applies to data-binding since it requires roughly the same kind of services.

When this is done, not only have we made Trident Core smaller and more manageable, but we can now apply more manpower to Trident Editing and Data Binding and extend the capabilities of these areas. In addition, this will showcase the use of the new APIs for other tools or applications to build upon.

Primary clients for the new Trident Editing component should be Outlook Express and Outlook. We may want to also retarget FrontPad to use this instead of FrontPage code and save a few Mb in the download size. VID might ultimately decide to use it as well - or rewrite its own. And other tools like FrontPage, Publisher, NetDocs or other part of Office will have the same option.

I consider this a very important task item for IE5. We should not, at all cost, continue to hack editing and databinding features in the current code base.

Tag Extensibility

If Trident is to be used as a universal run-time for UIs and Documents, and HTML as the language to describe them, we have to recognize that we will never be able to built in every single feature they need. We have to let third parties enrich both the HTML language and its implementation.

HTML is not very well suited for extensibility because of two major reasons: 1/ it lacks a mechanism to let people add new semantic without stepping on each other and 2/ two much specific knowledge on how to interpret it is built in into the engine, making true extensions difficult. XML however, with a pure syntax - that supersets correctly formed HTML - and the concept of name spaces seems better targeted at providing this extensibility.

Using the concept of name space, our parser should allow XML and HTML to be mixed in a single document, making XML a true and open extension language to HTML. In other words, the parser needs not only to be able to handle XML sections and parse them according to XML rules (stricter than HTML), but also support nested use of name spaces, including HTML inside XML. In addition we need to be able to parse and interpret XML schemas.

XML tags need to be treated like any HTML tags and participate in the Trident Object Model in the same manner. They should expose properties, methods and events like any HTML tag. They should support CSS styles (inline or global). Scoped XML tags should be able to contain text and affect the layout of that text through CSS.

But we need to go farther than just letting XML extensions participate into the object model and into the layout via CSS. Though this offers a first level of extensibility, it is not enough to let external groups such as Outlook, Office, DA/Chrome, Escher and others extend the browser capabilities with their specific features. DA/Chrome need to be able to build things like a "Path" element that move the HTML inside it or an "Effect" tag that operates on non rectangular areas. They are in fact already thinking about a complete XML description of any multimedia effects. Escher is doing the same thing. Word will want to add things like auto-numbered headings, annotations, revision control, endnotes, chapters. XL will need tags to describe formulas that hook up to their recalc engine through private mechanisms. Ultimately, us Trident, will want to add functionality without building more feature into the core Trident code base, such as more and cooler intrinsic controls for example.

This can't be achieved without associating *new behavior* to the new tags. Trident needs to allow external components to implement the behavior of certain elements in its HTML/XML tree, while continuing to consider these like native elements for parsing, object model, DHTML manipulation, editing, etc. This can be achieved by coupling internal nodes of the tree with externally provided COM or COR objects which Trident will instantiate based on the XML name space and name of the corresponding tags. The external object should be delegated some amount of behavior and should be able to participate to the object model, the formatting, to fire events, to send notification to other objects, etc.

Initially, these components will be able to implement behavior or influence the display via existing and new layers of APIs such as DHTML, the new low level editing layer, the drawing surface. In the future, they may directly participate to the layout, the display tree, etc.

These extensible elements need to be writable in C++ for the perf conscious such as DA, Escher, Office, etc., but also in Java, in HTML and Scripting or even in XSL. We need to watch out for the mistake made in OLE controls and not repeat them. In particular, it should be possible to write such a component with very little coding. In other words, no interface or method beyond IUnknown should be required. Trident should provide a default implementation for everything, therefore requiring the component developers to provide only deltas from the default behavior.

In the first version these extended elements may not be downloadable on demand (like activeX_controls and applets) but instead require explicit installation to minimize the security implications. In later versions, however, I would expect them to be downloadable by pages.

Other Architectural Enhancements

Elements vs. sites
Trident makes internally a strong distinction between "elements" and "sites". The former are things that are normally laid out by the text engine (B, I, P, etc.) while the later own a rectangular area of the page and usually provide their own rendering (IMG, APPLET, OBJECT, intrinsic controls, etc.).

This distinction made sense a year and a half ago when we started the design. However, since then, CSS introduced ways to morph a element into a site. Four CSS attributes have the ability to make any element turn into a site: float, width, height and positioned.

This has been a major road block toward implementing the CSS positioning spec in its full extent. We only support the positioning attributes on sites, DIV and SPAN. Supporting the full spec will require serious re-evaluation of the site vs. element architectural split. We should at least investigate.

Proxy-less (or rather, proxy-all) tree
The HTML tree currently contains two types of nodes: elements and proxies. This complication, though required to support overlapping HTML element, has generated much confusion and many coding errors and bugs in the past. EricVas has long proposed a rationalization of the model that would use only proxies in the tree and put the real elements on the side. This would streamline a lot of code and prevent numerous bugs in the future. It would also make the extensibility model described above easier to implement. We have been putting off this work in v1 but I think it's time to bite it off.

Windowless Trident
Currently Trident uses a window. This makes it non-transparent and prevent:

- To implement an equivalent of the Netscape LAYER tag (transparency required).
- To implement lighter-weight / faster frames, floating frames or desktop components.
- To let hosts use Trident as a display surface (problem for Chrome for example).

Making Trident windowless is quite a lot of work and would constitute the first implementation of a windowless OLE container. This may be too much work for IE5 but should be at least considered. We should at least make progress in this direction in the IE5 timeframe, i.e., depend less and less on the fact that we have a window.

Display tree
Our display engine consists of "Display" objects that each contain a list of "chunks" to be displayed on the screen. Some of these chunks contain text and some actually point back to an HTML element which either draw itself or points to another Display containing yet more chunks. This more or less forms a "display tree". However, it is not very well formalized as such. Too much of the display tree intersects with the HTML element tree - some nodes are common to both trees, some of the display data is in the HTML tree, etc. - making it difficult for example to create several display trees for multiple views. We should make steps toward fixing this.

The major advantages of a well architectured display tree are:

- Performance: parts of the tree can be optimized, merged, cached, etc., for optimal rendering.
- Extensibility: the display tree can be formalized and exposed to outside components for participation in the rendering.
- Multiple views: by creating several display trees.

This work - or at least part of it - is required if we are serious about improving our rendering performance for animated content.

Compositing engine
This basically consists in optimizations on the display tree, particularly caching of certain portions of it so that for example an animated or moving object can be efficiently rendered on top of a more static background. Again, this is an important part of getting Trident to render complex multimedia content with decent performance.

Multiple view support
This is another application of the display tree, though additional work is necessary generate multiple views and keep

them synchronized. Among the work needed:

- Reorganizing the top level Trident object (the "document") to split the non-view dependent data and the view dependent data into different objects, so that hosts can create and manage multiple instances of the later.
- Supporting multiple format caches per element (one per view).
- Supporting per-view style sheets and inline styles (in addition to the global ones).
- Modifying the current object model to pass view identifier whenever appropriate.
- Adding new view-specific object model to manage the views.

Multiple view is a critical feature for authoring tools (FrontPage already supports it), Office, NetDocs. Beyond the simple splitter bar feature that Excel supports, it can be used for draft, outline, printed page, slide view, etc., as do Word, Powerpoint and others. I am not convinced yet that we can actually implement this in its full glory in IE5 but we should at least take steps toward that goal in the architecture and cleanup work that we undertake.

Finish IE4

CSS support
At the very least we'll have to finish the Level 1 support:

- borders/padding/margins on phrase elements,
- consistent support of CSS attributes on intrinsic controls,
- control over element flow (inline, block, list-item),
- and a couple other features we missed in v1.

Possibly, we may have to support some of CSS Level 2. This will depend a lot on what Netscape implements and how much we want to push the CSS specification forward. Scottl and CWilso should give ideas on this.

We also need to get feedback on our CSS extensibility mechanism and see whether it needs to be enhanced further.

We need to complete the style sheet object model.

CSS Positioning
We score an incomplete on this for v1, mostly because of the number of bugs and glitches that we are going to ship with but also because of limited support in some areas.

The major hole is not being able to float and absolutely position any element (currently limited to sites, DIV and SPAN). We also need to support positioning of table components (rows, cells, etc.), as well as showing and hiding them.

DHTML Object model
Again here, v1 has limitations that author will have to work around. Some of the major ones are:

- Content manipulation properties (innerHTML, outerHTML, etc.) do not work on all element. For example they don't work on table elements, or in the head. This needs to be fixed.
- The text range is ambiguous with respect to HTML manipulation. We need to come up with a smarter range (high fidelity range) that allows non-ambiguous manipulations of the HTML.
- The table Object Model needs to be completed.
- Our event model is still to limited. For example, we don't expose events for certain actions (drag & drop, layout, rendering, etc.) and don't support bubbling of arbitrary events.
- We need to allow complete manipulation of the page: for example, give full control over head elements, make all collections read/write, morph elements to sites and non-positioned items to positioned (mentioned above already), morph between various INPUT control types, etc.

MWallent, RGardner and DanDu should complete this list. We also need to gather feedback from ICPs, ISPs, etc., and fill in every major hole they complain about.

Other Highly Leverage stuff we left out in v1

Among them:

- Add an HTML Text Area control.
- Complete HTML dialogs so that we can really position them as an alternative for Windows resource based dialogs.
- etc.

Trident vs. Browser

Our relationship with the Browser control (ShDocView) has grown more and more complex. The navigation interfaces are quite convoluted and the distribution of responsibility between Trident and the Browser has become obscure in places: for example some parts of the object model are under the responsibility of the Browser but still implement inside Trident code. We ought to look at this and ask ourselves whether it's justified or not.

In addition there are things that the separation between Trident and the Browser make difficult:

- The Browser adds its own window on top of Trident window, making it that much more difficult to implement a truly windowless browser, needed for implement layer-like functionality and to make frames lighter weight (and faster) than they are today.
- Because the Browser control also includes UI (menus and toolbars), it tends to depend a lot on Shell innovations, making it difficult to separate the Browser from the Shell and rev them separately.

In light of these considerations, we may want to investigate two directions of research:

- Separate the UI from the functionality in the Browser. In other words, implement a Browser control including navigation, history, favorites, object model, etc., but not UI. This may make it easier to break the dependency of the Browser on the Shell.
- Get rid of the DocObject hosting part of the Browser and use Trident as a doc object host instead (which it mostly is already). This is a more radical proposal, but it would remove quite a bit of code, get rid of one window for every browser or frame (and potentially two if we make Trident windowless as well), and streamline the distribution of the object model between Browser and Trident.

Note that these are just directions of research and may turn out to not yield any interesting results. But I feel that they are worth investigating as possible ways to de-couple Shell and Browser.

Competitive Response

Layer tag equivalent

As I mentioned above, this is difficult without making Trident windowless. We will have to assess how many people request this feature. Some ICPs have already asked for it. We will have to watch whether there is widespread demand for this. If yes, we may have to bite the bullet.

Note that Layers have a specific object model in Nav that we may or not have to be compatible with.

JSSS support

If this is approved by W3C in some sort of shape, we are likely to have to implement it. If not, we are probably off the hook.

Object model

Based on IE4 feedback, we may have to implement some of the Communicator/Netcaster object model that we have not implemented yet. We'll also have to match whatever Netscape come up with in the future version. Scottl should take a guess at this and provide a list.

Layout Enhancements

Architecture Enhancements

- We need to investigate whether integrating Line Services is a good thing or not. Will these let us achieve much higher typographic quality ? How will it affect performance ? How much work is it ? Based on answer to these questions we may or may not go ahead with the integration.
- The Text Selection Record needs serious rewrite if we want to support generalized focus on any element.

Typographic Enhancements

Many features have been suggested in this area. Among what could be considered:

- Pair kerning, character space control
- Better justification algorithm
- Hyphenation
- Better super/subscript support
- More decoration styles (more underline types, shadows, etc.)
- FE specific stuff

Layout enhancements

Most of these will be completion of CSS Level 1. For example, we need to support borders, margin and padding on phrase elements (Netscape already supports this, though in a buggy way).

Among the possible other layout features:

- Inside-out layout: the ability for the size of an inner element to control the size of its container.
- Better treatment of table headers and footers (scrolling tables for example)
- Snaking multi-columns
- End-notes, footnotes
- Netscape "Spacer" tag equivalent

Print / Print Preview

- Page layout features: section (chapters), per document or per section header, footer, page margins, gutters, finer control over page numbering, footnotes, etc.
- Show text as printed (ability to layout to the printer device while rendering to the screen for page view / print preview). This is something that all good word processors do very well.
- Print preview UI. This should be implemented outside of Trident Core.

Editing Services

New low level editing services API
This is the pre-requisite for any editing improvement. I described it at the start of this memo. The primary idea is to expose a set of object model allowing precise manipulation of the HTML tree.

Better user input event hooks
In order for Trident to be a multi-purpose editing surface that tools and authoring application can build upon, we need to surface more events and let external components that hook these events override all Trident default behavior. Among the needed events are raw user input event (mouse and keyboard), drag & drop events, hooks for background proofing tools or to implement URL recognition.

The selection model needs to be extensible so that Editing Components can provide different or enriched selection model, while still making them accessible through the object model. The canonical example is the table selection or the object selection model which should be implementable outside of the core Trident. The Editing Component should be able to control the visuals of the selection model (for example the shape and colors of selection borders and drag handles) at least to some extent.

Intrinsic controls

If we are serious about Weblications, we need to provide authors with a rich set of intrinsic controls. They need to be:

- Cooler: we need more and cooler styles of controls (nice picture buttons, different border types, background textures, etc.)
- Data-aware: some of them already are, but we are missing at least a data-bound listbox.
- More powerful: rich text box, auto-complete in listbox, etc.

Windowless, richer and databound SELECT

Our SELECT control (listbox and combo-box) could use several enhancements, the main of them being to make it windowless. It is currently based on the USER control and this has caused us much difficulty to make it support CSS styles. In addition, having a window prevents it from being transparent and complicates our positioning and scrolling code.

An alternative and richer approach would be to build the SELECT control as an HTML page. Basically, an HTML page would describe the layout of the list (for example as a table). The SELECT control would simply display this page and implement a list-like selection model on top of it. This would allow much richer and cooler looking lists, multi-column lists and data-bound lists by simply using databound tables.

HTML TextArea
This is something we wished we had done in IE4 given how close we are. Our TextArea control is based on the Trident text engine and as a result is perfectly capable of handling HTML instead of plain text. This is something we absolutely need to enable in IE5. It's highly leverage and has tons of user benefits, for example for Web pages that are collecting user feedback and mailing a form back to the site. Rich text would be goodness in such a situation.

Improved CSS support
Our intrinsic controls have somewhat limited CSS support. Not all CSS attributes are supported, especially for the SELECT control. Fixing this is part of making the intrinsic control cooler and more powerful.

Page selector control
Many sites on the Web use some sort of graphic or image map at the top of on the left to switch between pages. In most cases, currently, this involves server round trip which is slow and ugly. As DHTML becomes more popular, people will start building the same effect by switching DIV visibility on the client. All of this is possible in IE4, however there is no real Page Selector control (sometime called Tab control) as VB or Office have. In addition such a control is needed for HTML dialogs.

Building in such a control would add a lot of value. It needs to be extremely cool: accept all sort of coloring and bitmaps for the tabs, make it easy super easy to implement transition effects, highlighting effect, sound effect when fl:pping pages, etc.

OLE Control hosting
COM & COR component hosting

Our OLE control hosting support is pretty extensive. In a sense I would be tempted to say we are done in this area. However, there are a few omissions in IE4 that we should look into:

- Better design time support: we leave most controls not-inplace active at design time, which not only make them totally dead but for most ends up displaying a place-holder since they don't support drawing in a non inplace state. We need to fix that if we are serious about support Weblication authoring tools.
- Support for Design Time Controls (DTC). These are controls that are specific to design time and generate different HTML at runtime. VID support them by pre-parsing and post-parsing/modifying the HTML that they hand us. We may want to consider native support

Finally, if COR becomes prevalent in the IE5 time frame, we may have to start investigating support for COR components / controls, and possibly support for IronWood components.

Multimedia enhancements

In general, I believe our approach to multimedia effects should be to provide the support and the core technologies needed by other groups (DA/Chrome, Escher, etc.) to implement their features and recommend that they build on top of our extensibility mechanisms. However, there may be certain features that we consider required for IE5, whether we implement them ourselves or make sure that these outside components have them on their plans.

Among some of the supporting features that we can consider for DA/Chrome:

- Tag extensibility (see above)
- Display / compositing engine enhancements (see above)
- Make more Trident components and Trident itself windowless (see above)
- Enhance the filter architecture to let external components apply filter on certain areas of the HTML but lay these out differently (for example so that Chrome can put a DIV on each side of a cube).

This is very incomplete. FrankMan, DonMarsh, MichaelW have ideas on all of this and should put a list together.

Weblication support

Most of the above will benefit Weblications. However there are also a few other direction of research more specific to Weblication:

- **Data**: I am still putting together the data on this. Need more discussion with JeanPa, AdamB, etc., to understand how much support is needed in Trident for the data pump and various binding / notification mechanisms.
- **Root document / Shared & persisted state**: we will need to expose some way for multiple pages to share state and / or persist state. Not sure yet whether this should be done via the concept of a root document (and possible hierarchy underneath it), a glorified cookie mechanism, or by sharing parts of the HTML on each page ala WalkAbout. This is an area that upcoming discussion on Weblications will hopefully shed some light on.
- **Control over object lifetime**: the lifetime of a page and all objects it contains is currently limited to the amount of time the user is looking at it. There are many cases where authors need to be able to extend and / or control the lifetime of a page or some of its objects in order to retain or share state as the use navigates. We need to investigate ways to do this.
- **Control over UI**: pages in a Weblication will need to alter the browsing UI. It is not obvious whether they would need to control the toolbar and menus since these are outside the real estate of the page, but they certainly need control over context menus. Our object model should allow HTML elements to participate into the context menus so that

different areas of the page can define different context menus. This would potentially be done through CSS attributes.

This is still very incomplete. The upcoming discussions on Weblication should clarify and complete it.

Performance

The download performance is now pretty much as good as it will ever be. However, there are still many areas of improvement for Trident, particularly in the layout engine, large tables, editing performance.

Most critical performance tasks are:

- Improve the rendering/compositing engine.
- Reduce the number of Monster Walks (the mechanism that updates all caches when a tree change occurs), improve handling of text changes in the text change queue.
- Improve script executing speed.
- Improve editing speed.

Globalization

Far-east layout features

Vertical text
This section to be filled in later

Other misc stuff

Sending charset on Submit, specify charset on save.

Data-binding Component

This section to be filled in soon

Trident Editing Component

This section to be filled in soon.

General architecture

Enhanced text editing

Table editing

Object manipulation

2D/Positioning objects design surface

CSS editor

Script editing / Debugging

Other misc editing

User Interface elements