Level: Intermediate
Works with: iNotes Web Access
Updated: 01-May-2002

by Vinod Seraphin
with David DeJean

Dynamic HTML (DHTML) is not just one technology; it's a combination of several Web technologies. And this combination has one great benefit for developers: It helps you create the best possible user interface for your Web applications.

We used DHTML to create a dynamic user interface for iNotes Web Access—and learned a lot along the way. In this article, I'll explain some of the methods and techniques we used when creating iNotes Web Access. This article assumes experience as a Notes/Domino applications developer with some knowledge of HTML and JavaScript. You can download code examples from the **Sandbox**. And if you want to take a look at iNotes Web Access (and kick its tires), go to the **iNotes Web Access live demo** here on LDD.

A true DHTML story

When I started at Lotus, I worked on Lotus Organizer, a program that was famous for its great user interface. Since Notes R5 shipped, I have spent a considerable amount of time learning as much as I can about the DHTML capabilities of Internet Explorer. The more I learned about DHTML, the more convinced I became that much of what others were trying to do with Java applets in a browser could be done with DHTML. This lead to the creation of iNotes Web Access, for which I am presently the lead architect.

The first version of iNotes Web Access shipped in July of 2001as part of Domino 5.0.8; the second major version will ship with Notes/Domino 6. I think the success of the first version shows what you can do with DHTML. Many folks found it hard to believe that we weren't running extensive Java applets or ActiveX controls. But it is true that iNotes Web Access requires nothing more on the client than the Internet Explorer (IE) 5 browser, and that most of its cool features are implemented using DHTML.

Web applications haven't been celebrated for their interactivity, and that's understandable. Browsers have supported only limited programmability—too often in nonstandard ways. The standard programming language, HTML, was developed to format text, not support user interaction with data and the browser's interaction with the server. Filling out a form and clicking a button labeled Submit was about as interactive as the Web got in its early days.

Things have improved greatly, of course. Successive versions of HTML have methodically enhanced its support for interaction between the user and the data. The increasingly powerful scripting through JavaScript or VBScript in the browser opens up the processing power of the client to developers. The Document Object Model (DOM) for HTML and Cascading Style Sheets (CSS) make it possible to not just manually mark up a page but to control its appearance and function programmatically.

By using these Web technologies and some of the same design approaches we took in iNotes, you can make your Web applications more dynamic, too. If you've seen the presentations Dan Gurney, a colleague on the iNotes team, and I have given at Lotusphere or Devcon and downloaded our examples from the **Sandbox** you're

acquainted with three powerful ways to exploit DHTML:

- Using scriptable events
  In IE, every HTML element can be addressed by ID. You can detect user actions by writing event handler functions and hooking them up to various HTML elements, which in turn, can cause the page to dynamically change as the user interacts with it.
- Hiding and showing objects
  CSS supports positioning objects and elements on the visible page, or even away from it, in invisible planes. Later in this article we'll look at the datepicker control in iNotes, which creates a drop-down calendar in an invisible plane, and then reveals it under user control.
- Changing the appearance of objects
  Style attributes can be changed dynamically using JavaScript hooked to user actions through events in the browser. Again, we'll look at examples from the datepicker.

You can deliver these basic DHTML development strategies in a variety of ways. You'll use the same elements whether you're serving up DHTML using passthru HTML in Domino applications, static files, JSP or ASP pages, or even turning it into a tag library for JSPs. However your DHTML is delivered, it should make your applications more usable, and it should let you code in building blocks that make it easier for you to put together pages.

What is DHTML?

Dynamic HTML is a label that ties together several distinct Web technologies. These include HTML 4.0, the browser's Document Object Model (DOM), Cascading Style Sheets (CSS), and a scripting language.

HTML 4.0

The Hypertext Markup Language (HTML) was originally developed as a way to let nonspecialists format documents for publication on the World Wide Web. Its tag-based structure was easy to learn and apply, but the result was static documents that mixed formatting and content. Pieces of content within the documents weren't identifiable as separate elements and couldn't be reformatted dynamically. The standards for HTML, which are set by the World Wide Web Consortium, have changed a great deal over time. The 4.0 version of the HTML specification is the most recent major version. It adds addressability for content elements; elements may have an ID or CLASS attributes (or both). The formatting can be set in a style definition, either in the same document or in a separate document called a style sheet.

The DOM

If you've written any LotusScript you're familiar with the basic concepts of a Document Object Model, or DOM. In a Web browser, the DOM treats each HTML element on the page as an object and provides ways to set and get various properties and invoke methods associated with the element. These objects also fire various events as a user interacts with them.

One of the challenges of coding DHTML is that the DOMs of the two most widely used browsers, Internet Explorer and Netscape Navigator, are at times, incompatible. When we started developing iNotes Web Access, IE 5 provided a DOM vastly superior to Netscape 4.x, which is why iNotes was first available for IE. (See more on this in the **Internet Explorer and/or Netscape?** section below.)

Cascading Style Sheets

The Cascading Style Sheet (CSS) is another standard established by the World Wide Web Consortium. Like HTML, CSS has evolved over time.

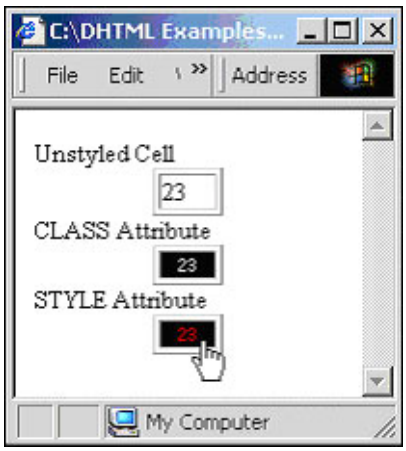
- CSS1 was the first specification. It introduced rules that may be applied to HTML elements to set properties such as color, alignment, border styles and widths, margins, padding, and much more. You can apply these rules to individual elements identified by ID attributes, to all HTML elements of a kind (such as <p> elements), to all the members of a group created by a CLASS attribute, or to sets of elements combined into an object by a tag.
- CSS-P extended CSS1 syntax with rules that allow absolute or relative positioning of the object. You can specify an exact pixel location on the screen, or position the object off the screen, on a transparent layer "above" the visible screen. Again, there are incompatibilities between browsers. IE allows positioning of any visible HTML element. Netscape 4.x allows positioning only of a <layer> tag and a LAYER object that can be addressed by script. The <layer> tag may be considered a means of delimiting and containing a more complex set of HTML. The tag, not supported by Netscape 4.x, is the way to do this in an HTML 4.0-compliant manner.
- CSS2 incorporates rules introduced by CSS-P and adds some more.

Let's look at an example of a simple CSS style definition. CSS introduces an HTML STYLE element, which has start and end tags and allows style rules to be specified within the STYLE block (The HTML start and end

comment tags within the STYLE block are there so that older browsers will ignore what's in-between). Here's an example of a rule (in this case a class definition) with various properties and values:

```
<style type="text/css">
<!--
.CurrentDay {background-color:black; font-size:7pt;
font-family:Arial, Helvetica; font-weight:bold;
color:white; text-align:center; cursor:hand}
-->
</style>

Unstyled Cell

<center>
<table width=36 border=1>
<tr>
<td>23</td>
</tr>
</table>
</center>

CLASS Attribute

<center>
<table width=36 border=1>
<tr>
<td class="CurrentDay">23</td>
</tr>
</table>
</center>

STYLE Attribute

<center>
<table width=36 border=1>
<tr>
<td class="CurrentDay" style="color:red">23</td>
</tr>
</table>
</center>
```

Here's what this code looks like on the screen:



You can see that styles give you much more control over the formatting of objects—you can specify the font size precisely in points, for example, rather than just in the relative size supported by HTML. You also get control over some UI elements like the cursor shape. The STYLE attribute may be applied to any element either by itself or in

conjunction with the CLASS attribute; the example uses a STYLE attribute to override the class definition's attribute for the font color, for example.

Scripting languages
A scripting language is used to manipulate the DOM and handle events fired by objects. JavaScript is the primary scripting language used in Web pages and fortunately this is one area where browser incompatibilities are minimal. Microsoft created its own dialect of JavaScript, called JScript, for IE. The international standards body ECMA produced a harmonized specification, called ECMAScript, which was adopted by both Microsoft and Netscape, so while you may occasionally see the name ECMAScript, it's really just another name for JavaScript. Microsoft continues to support JScript in IE, and also supports Visual Basic as an alternative and provides mechanisms to add support for other languages. We will deal only with JavaScript.

Internet Explorer and/or Netscape?

When we were developing iNotes Web Access, the goal was to fully exploit the next generation of browsers. They offered tremendous improvements in programmability and dynamic page update capablities, which in turn, allowed for a much better user experience. We did not want to build a "lowest common denominator" solution that would work on all browsers. We settled on Internet Explorer 5 as a good example of a next generation browser, while also developing a version of iNotes for Netscape 4.7.

These two browsers required very different code (as you'll see if you look at the code samples). When iNotes was shipped in July of 2001, the Netscape code was blocked and IE was the only supported browser. Time and resource constraints played a part in this. Developing some of the functionality we wanted also turned out to be much more difficult for Netscape due to a lack of a quality debugger for this environment. (Incidently, the next version of iNotes, which will ship with Notes/Domino 6, will support Netscape 4.7 as well as IE.)

Important scripting features
IE supports a rich DOM that included features we needed to create a better user experience. In the IE DOM, you can assign an ID attribute to any HTML element. This is actually part of what is known as DOM “Level 0” (what was in the level 3 browsers). When it has an ID, any element can be addressed by script. IE also supports a special “all” collection that includes all HTML elements within the document that have an ID attribute assigned. The IE DOM also makes available a rich set of properties and methods for any element, making each element much more programmable.

Also, in IE objects are available immediately after they are written to the page within a script block or within static HTML. With Netscape, the few objects that are programmable aren’t available until the ONLOAD event.

IE supports many more events on more elements than other browsers. For example, in IE, any visible element can support the onclick event. In Netscape 4, this is limited to very few HTML elements such as INPUT elements and Anchors. IE also supports what it calls “event-bubbling.” This refers to events being passed on to the element’s parent element after the original element has had a chance to react to the event. This makes it much easier to write code that supports special processing. Take a calendar created as a table with a cell for each day. Without event-bubbling, making something happen when the user clicks on a particular day cell means that each day cell’s date text would have to be enclosed within an anchor tag. In IE, the same behavior can be implemented without any anchor tags, simply by defining a single onclick handler for the table as a whole. This greatly simplifies the number of elements needed and the size of the HTML you need to generate via script.

Lastly, IE 5 introduces something called expressions that, when used in conjunction with a CSS style, allow one style property to be dynamically updated when another property changes. For example, expressions can be used to resize other elements on the page when the window itself is resized.

Dynamic page updates for interactive applications
What really sets IE apart from other browsers is its dynamic page update: it allows you to update any element on the page without reloading the entire page. It does this by giving every HTML element on the page a set of read/write properties that allow modifying either the element or just its contents:

- innerText and innerHTML allow access to any text or HTML contained within the element (in other words between the element’s start and end tag).
- outerText and outerHTML allow access to any text or HTML including the element’s start and end tag. This provides a means of modifying the elements attributes.

In addition to these powerful properties that can be used to modify the element itself, IE also supports methods to manipulate the text or HTML associated with an element. These include insertAdjacentHTML, insertAdjacentText, getAttribute, setAttribute, and removeAttribute.

Dynamic page update makes it very easy to script an event that changes an element or object, and then give up control by returning from the event being handled. IE will automatically update the screen to reflect the changes made by your script smoothly, without redrawing the entire screen.

When we were writing the first version of iNotes Web Access, IE was the only browser with this dynamic update capability. The good news is that Netscape 6 now also offers similar capabilities (and even the innerHTML property), and so will all future browsers.

Key building blocks of a dynamic UI

I mentioned three key building blocks of a DHTML user interface—positioning elements, hiding and revealing them, and handling events. We'll look at code examples taken from iNotes Web Access and discuss the features in HTML, CSS, and the IE DOM that make them work. Before we start, let's review some of these that are very basic, like the and tags and attributes of the style.

Positioning elements

If you've written any HTML, you know that the <table> element is the best tool for positioning content elements on a page—in fact, in the early versions of the HTML spec, it was about the only one. DOM Level 0 added two important HTML tags, and . You'll still create lots of tables, but and considerably broaden your options for positioning and styling page content.

- The element is recommended for any block content—elements and objects that will occupy a rectangular area of the screen. Typically, content enclosed in a tag starts at the left margin and occupies a specified width and depth. The element that follows the end of the block is automatically placed on a new line following the block.
- The element is for any in-line content—a contiguous string of text and/or HTML elements. The element that follows a span is not automatically placed on a new line.

By specifying the style attribute for position and coordinates, you can place a block where you'd like on the screen. The CSS-P positioning attributes also let you set width and height in a variety of units—not only pixels, but inches, picas, and ems (a measurement related to the element's font size) among others. Here's an example of a element that would position whatever content it enclosed 30 pixels from the left edge of the window and 40 pixels down from the top, and make them rectangles 100 pixels wide and 50 high:

```
<div style="position:absolute; left:30px; top:40px; height:50px; width:100px">
....
</div>
```

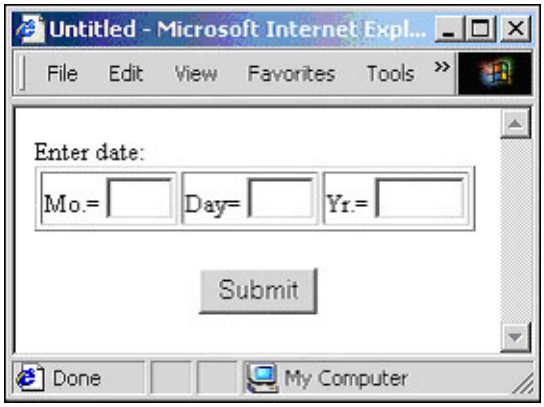
Hiding and revealing elements

Elements are hidden or revealed by using either the VISIBILITY or DISPLAY attributes.

- VISIBILITY can be set to "hidden" to remove an element from the page display, but it preserves the space the element would occupy if it were visible. Set it to "visible" to reveal the element.
- DISPLAY, when set to "none," hides the element from view and closes up the space it would occupy. Set it to "block" to reveal.

Creating a great UI

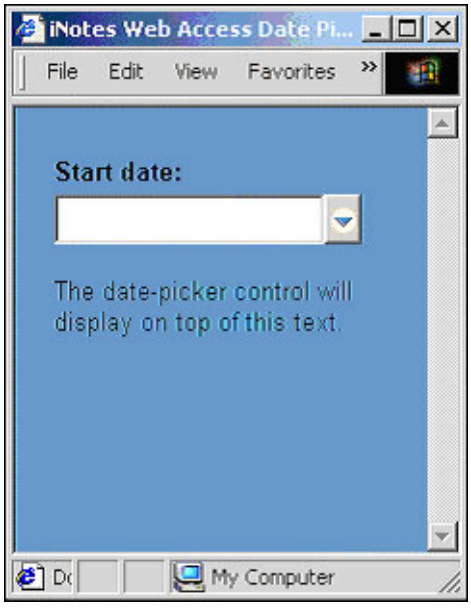
The measure of a great graphical UI is how easy it makes it for a user to quickly and accurately perform the tasks required by the application. Take specifying a date as an example. The UI could be as simple as a text box for the user to type a date into. Here's a typical HTML-only solution:



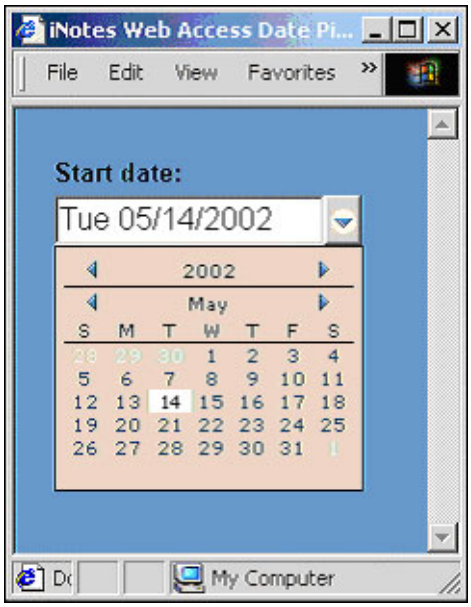
This doesn't give the user much help. In particular, it doesn't help the accuracy of the entry. The user could easily mistype the date or enter it in the wrong format. The multiple boxes give the user some indication of the data and format. But how does the user select a date? If she wants to choose the second Tuesday in the month, she has to turn away from the application, perhaps even leave her computer to find a calendar.

In contrast, let's take a look at a date control derived from iNotes Web Access. Dan Gurney and I used this example in our session on DHTML at Devcon 2001, and you can download the source code from the **Sandbox**. The examples include many sample DHTML applications for tasks like scripting events, creating and populating a table, and manipulating calendars.

The date control is in the sample file named Demo_DateControl.html. This control gives users the kind of help they need when entering dates. It begins with two visible elements, a text entry control and a button:



Clicking the drop-down arrow drops down a month calendar and automatically inserts the current date into the text field:



The user can choose another date by clicking it (if it's in the current month) or clicking the navigation arrows to traverse the months and years at the top of the calendar panel. Selecting a specific date or clicking outside the calendar panel will dismiss (hide) the panel.

This UI has obvious advantages over the HTML-only solution. The drop-down calendar panel gives the user all the information they need to select a date in the familiar visual metaphor of a monthly calendar. It provides an intuitive way of navigating that information. It also ensures the correct format and accuracy of the entry, because all the data is generated by the application, not the user.

You can probably already begin to see how this was created in DHTML:

- The calendar panel is a table within a block that is hidden until code hooked to the button's onclick event reveals it.
- It uses classes and styles to format the individual date cells, and onclick events to restyle the selected cell and transfer the date to the text control.
- IE's dynamic update lets the calendar update smoothly as the user clicks the navigation buttons: innerHTML and innerText attributes are used to change the cell contents—there's no Submit button, no screen repaint.

The basics

The source code for the page is short—only 119 lines—but that's because the heavy lifting is done in four related documents, a style sheet and three JavaScript documents, that are included in Demo_DateControl.html by reference:

```
<link   rel="stylesheet" type="text/css" href="../common/StyleSheet.css">
<script language="JavaScript" src="../common/iwa_DateTime.js"></script>
<script language="JavaScript" src="../common/iwa_DatePick.js"></script>
<script language="JavaScript" src="../common/iwa_Utilities.js"></script>
<script language="JavaScript" src="../common/iwa_DropDownMgr.js"></script>
```

The segmentation of the code into multiple documents helps make it reusable. The file iwa_DropDownMgr.js, for example, creates a JavaScript object to manage any panel (… block) to be dropped down near an <input> element. The contents of the block could be anything. Here, the contents are two tables that make up the datepicker. The same code could be used elsewhere in the application to display completely different content—a list, for example.

The style sheet, StyleSheet.css, sets default styles for the various classes of elements in the datepicker. These are assigned dynamically as the datepicker code builds the calendar, and may be dynamically reassigned by code linked to event handlers. For instance, the day style for most of the day cells in the calendar table sets the background color to a light olive. The style for the selected date uses a black background. One of the actions tied to the onclick event changes the style applied to the selected cell.

The file iwa_DateTime.js is a collection of functions to do date management. It performs tasks like the validation of dates entered from the keyboard and the comparison of two dates. It extends the JavaScript DATE object. (In Organizer, we had used an AdjustDate function similar to the @Adjust function in Notes. We added similar functionality in JavaScript.)

The iwa_DatePick.js file is the code that builds the datepicker, sets today's date, and styles the content. The datepicker is two stacked tables—one for the year and month navigation a the top of the panel, and the other for the week rows and day cells at the bottom.

The file iwa_Utilities.js contains frequently used functions to do basic actions like get the absolute position of an element on the screen, or chain together several function calls to an event.

Scripting events
The onclick event is used to change the calendar display by month or by year. It does this by adding an attribute, to the <img> tag for the arrow graphics that the user clicks to change the displayed calendar. Here's the tag for the "Previous Year" arrow. The attribute YEARADJ is assigned a value of -1:

```
<img class="s-datePick-Arrow" src="../common/DNPrev.gif" alt="' + "Previous year" +"' border=0 YEARADJ="-1"
/>
```

Note: Another neat feature in IE 5 is that developers may add any additional attributes to an element by simply defining an attribute, as we have done above. This then results in this attribute being available off the element's object as well.

The onclick event handler for the element is one of several event handlers assigned to functions as part of the datepicker construction in iwa_DatePick.js:

```
var el = aAll[this.id];
        . . . {
            el.oDatePick = this;
            el.onclick = DatePickOnClick;
            el.onkeydown = DatePickOnKeyDown;
            el.onmousedown = DatePickOnMouseDown;
            el.onmouseup = DatePickOnMouseUp;
            el.onlosecapture = DatePickOnLoseCapture;
```

The function DatePickOnClick validates the input, then passes the element name and event on to functions that use the element name (the image filename) and the adjustment value to determine what action to take, and then rewrites the calendar and updates the screen.

Hiding and revealing
The onclick event is also used to display the drop-down datepicker panel. The datepicker doesn't actually "drop down" from the input field when the button is clicked. Its VISIBILITY property is changed to make it visible. This is begun with the onclick event for the button in Demo_DateControl.html:

```
<input type=button class="s-dropdown-btn" onclick="showDatePicker('StartDate',this)">
```

The showDatePicker function takes two arguments, the name of the input field, StartDate, and the button element itself, referenced by "this." It does some validation—it checks the browser type and does some validation to make sure that the default date and the input field are both valid. Then it passes control to the DropDownManager object's show method. The method calculates the screen position for the drop-down based on the position of the calling element (the getAbsPos function comes from the iwa_Utilities.js document), and it sets the visibility attribute of all the style definitions to "visible":

```
    var pos = getAbsPos( el );
    var iLeft = Math.max(4, (pos.x + el.offsetWidth - elDD.offsetWidth));
    elDD.runtimeStyle.pixelLeft = iLeft;
    elDD.runtimeStyle.pixelTop = pos.y + el.offsetHeight;
    elDD.runtimeStyle.visibility = "visible";
}
```

("runtimeStyle" is a special property IE supports to facilitate the temporary adjustment of a style value. It come in very handy if you want to override a style without forgetting what the original style was, allowing the override to be

easily removed.) Then it displays the datepicker panel:

```
if (oDate) window.dateControl.setExternalDate( oDate );

// "drop" the date-picker
window.ddMgr.show(oBtn, window.dateControl, sFldName);
```

Changing the appearance of objects
The datepicker does just about the same amount of work to change the appearance of a selected date as it does to change the calendar to a new month: it calculates what action the user requested, performs it, and then regenerates the HTML for the entire calendar display. This may seem like overkill, but from a programming perspective, it was simpler to do this than to worry about changing the class associated with several cells on the screen. However, another approach—remembering what cells had special styles associated with them (“Today” and “CurrDay”), and then programmatically changing the class attribute of these affected cells—would have accomplished the same end result.

If you look at the iwa_DatePick.js script, you'll see that DatePick's update method writes the calendar table out as a text string, concatenating it in a variable named s. It picks up the values and attributes of each cell as it loops through the table. One of the ways it builds that string is by assigning string values to variables, and then assigning those variables to the innerHTML values of the <td> tags:

```
el  = aAll[ "SDP" + 'Grid' + this.sUnique ];
    str = '<table border="0" cellspacing="0" cellpadding="0" width="100%">' + this.s + '</table>';
    if (this.bHasIEDOM)
        el.innerHTML = str;
    else {
        el.firstChild.nodeValue = str;
    }
```

The innerHTML attribute, remember, is only what is between the opening and closing tags, not the tags themselves. (The reason the assignment isn't done directly [el.innerHTML='<"table border="0" cellspacing=. . ."] is the conditional, which in this case is checking to make sure that the target browser is IE.)

Summary: How "dynamicness" can help

The datepicker control is a good example of the UI work that's gone into iNotes Web Access. It shows its dynamic functionality, and admittedly, it shows its complexity—it's not easy to follow some of the examples through the JavaScript code.

In a way, though, this is a point worth making in itself: DHTML is a development environment that is fully capable of supporting complexity in the applications you develop. We developed iNotes to run in IE 5 by using the existing controls where we could and extending the existing ones or created our own when we had to. And the results were very good—Network Computing thought so, because it named iNotes Web Access "Best Web-Based Application" last year.

DHTML gives you much more control over the browser environment, which means it gives you the tools to develop much richer user interfaces. You can set ALT text to display as the cursor hovers over a feature (equivalent to Windows tool tips). You can control the shape of the cursor to indicate possible actions. You can create tabbed panel to indicate the variety of an applications functions or features. The options go on and on. And because DHTML executes in the browser, you can create these richer interfaces without negatively impacting server performance.

The real point, of course, is that although your application may actually be very complex, the interface you provide the user should be as simple as possible. Only then will computers truly help users be more productive. DHTML can help you build interfaces that make this happen.

ABOUT THE AUTHOR
Vinod Seraphin is a Senior Technical Staff Member and lead architect for iNotes Web Access. iNotes Web Access was "born" from Vinod's prototyping efforts to develop a very compelling Personal Information Manager (PIM) within a browser. Vinod was recently recognized with the IBM Outstanding Technical Achievement Award for his significant role on this project. He has been with Lotus/Iris/IBM for over 10 years. Prior to iNotes Web Access, Vinod was the Software Architect for Lotus Organizer. He has an MS in Computer Information Systems from Boston University and a BS in Computer Science and Engineering from MIT. He also is an avid professional sports fan and enjoys softball, skiing, traveling, and spending quality time with his family.

