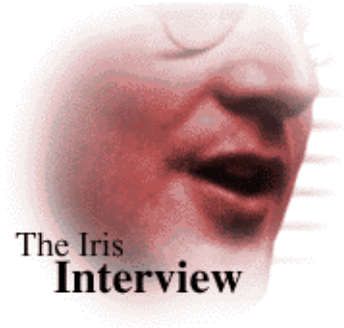


Mark Judd: JavaScript Integration

Interview by
Susan
Florio

Level: Intermediate
Works with: Designer 5.0
Updated: 05/03/99

Inside this article:

Related links:
The IDE & more: JavaScript support in Designer R5

Query Builder Example sidebar

Domino Designer R5 Technical Overview

Ned Batchelder: Designing Domino Designer

R5 Reviewer's Guide

Get the PDF:

MJudd.PDF70Kb

*[Editor's Note: To learn more about JavaScript in R5, check out the discussion with Mark in the **Developer Spotlight**. For specific information on using JavaScript and the new IDE in R5, see "**The IDE & more: JavaScript support in Designer R5**."]*

Meet Mark Judd. His motto: "Bring the Web to Notes and Notes to the Web!" His goal for R5: Integrating JavaScript into Notes. Here we learn how creating Web applications just became a whole lot easier!

What are the goals for programmability, and specifically JavaScript, in R5?
Well, our goal for R5 was to blur the distinction between a Notes client and a browser client. Anything you can do in a Web application, you should be able to do in a Notes application, and vice versa. To do this, we needed to make the client-side programmability story the same in Notes as it is on the Web. So, since JavaScript is essentially the scripting lingua franca on the Web, we brought JavaScript to Notes.

The long-term goal is to "write once, run everywhere" -- one application for multiple clients. Notes developers should be able to write one template that runs equally as well in a Notes client as it does in a browser. R5 gives us a good start down this path. In the future, the path includes bringing all the client-side processing that occurs in Web browsers, including the ability to manipulate an HTML page through JavaScript, and the ability to manipulate XML data through JavaScript, the ability to format data through XSL -- all of the Web technology that is out there -- into Notes. And we'll continue to pursue this goal in future releases.

Why is JavaScript so important to browsers?
It's important to off-load as much processing as possible from the server onto the client. Once your data is on the client, once a document loads from a server, you should strive to keep as much processing of this data as possible on the client. JavaScript allows you to create highly interactive applications that do a lot of client-side processing. The amount of network traffic is reduced when this happens, and the server is free to deal with other requests.

What version of JavaScript does R5 support and what does this mean

for users?
People ask this all the time, but it depends on what you mean. Do you mean the language, or the Document Object Model (DOM), or the classes, or the libraries? If you are talking about the language, Domino supports JavaScript Version 1.3. We've licensed the Netscape JavaScript interpreter and embedded it into the Designer and the client. For users, this defines the language syntax and the set of built-in classes, such as String and Date classes.

A side point about the language is that JavaScript truly is an object-oriented language. It's "object-orientedness," however, is based on a "prototypical instance" model rather than a "class" model. This instance-based model can be very powerful. But let's save this discussion for another day. What people usually mean when they ask "What version do we support?" is what do we support in terms of the DOM.

What is the DOM?
The DOM stands for the Document Object Model. When a browser renders an HTML page, it first parses the HTML and creates an in-memory tree representation of that page. That tree and the API that accesses and massages the tree once it is created are called the DOM.

The DOM states exactly what the objects are in the model. It also states the objects that can "contain" certain objects, and each object's list of methods, properties, and events. It states, for example, that a document contains forms, that a form contains elements, and that a text element has a "value." In JavaScript, you'd access a text field (or element) value using the following syntax:

```
document.forms[0].textFieldName.value
```

The bad news is that different browsers support different DOMs. Netscape does it differently from Microsoft Internet Explorer. Furthermore, 3.0 versions of the browsers support different DOMs than the 4.0 versions, which support different DOMs than the 5.0 versions. The Notes client will only add to the confusion, since we only provide a subset of the HTML 4.0 DOM. Our current DOM support is closer to the 3.2 definition.

The good news is that there is a W3C effort to standardize the DOM. We are committed to support their definition. To me, the most important aspect in the browser wars is not about the level of HTML support, but rather about the level of DOM support. I'm always looking for the most "programmable" DOM. Today, IE wins hands down, but watch out for Netscape's newest renderer (named Gecko).

What is the relationship between Dynamic HTML and JavaScript?
Dynamic HTML is a whole lot of things. Its major components are the DOM, Cascading Style Sheets (CSS), and JavaScript. But it's the DOM that really makes DHTML dynamic -- the fact that an API exists to manipulate a page after it's been rendered or even to create the page initially. Presently, the only ubiquitous language mapping for the DOM is JavaScript. Look for the Java language bindings to crop up in future browsers and in the Notes client as well.

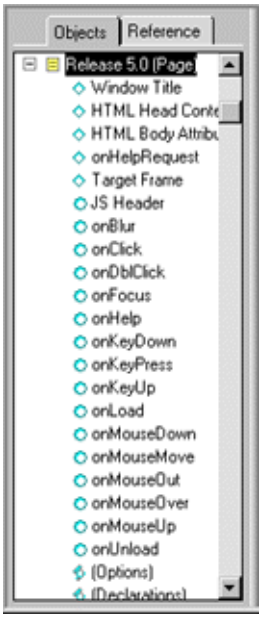
What are the most common uses of JavaScript in a Domino application?
In Domino Designer, you can use JavaScript to program actions, buttons, hotspots, and form, page, and field events. You'll often see applications taking advantage of these objects to perform form and field validation, to create mouse-over effects, do numeric calculations, and display dialog boxes.

What changed in the Designer between R4.6 and R5?
With R5, you can program JavaScript in the integrated development environment (IDE). Now, when you design an application, you can write

JavaScript directly without having to use pass-thru HTML as you did in R4.6. Remember, pass-thru HTML is only processed by browsers. JavaScript written in the IDE, on the other hand, is accessed both by browsers and Notes clients.

The objects in the DOM are exposed through the Programmer's Pane. In R4.x, when you created a field on a form, the field object appeared in the InfoList with its corresponding events (Entering and Exiting were supported). Now, when you create a field on a form, you'll see the JavaScript events that correspond to the appropriate DOM object as well. As you create objects on a form that correspond to DOM objects, you'll see the DOM events listed beside the NotesUI events.

The events that you can attach JavaScript to vary, depending on the object you are working on. To see which events are available for which objects, simply browse through the objects exposed in the InfoList pane under the Objects tab.



Also, when you are programming an event handler in JavaScript, the Reference tab in the InfoList will list the DOM objects along with their properties, methods, and events.

Is R5 HTML 4.0 compliant?
In the Designer, you can create Web applications that take full advantage of the HTML 4.0 browsers. The Domino Web server can serve up these 4.0 compliant pages. The Notes client does not yet handle onMouse or onKey events, so it is not currently 4.0 compliant. If you want to create a single application that runs in both a Web browser and the Notes client, you'll want to program to the HTML 3.2 specification.

However, even though onMouse event handlers written in JavaScript are not processed in the Notes client, you can still program certain mouse effects. The Designer allows you to program the "normal," "mouse over," and "pressed" states of outline entries, action buttons, and image resources without forcing you to program these state changes in JavaScript. The Notes client honors these state changes, making the need for onMouse event support less important.

Will it be a lot of work for users to upgrade?
In R5, you can run your R4.6 applications as they are coded and they will work fine. However, as I said earlier, pass-thru HTML and JavaScript aren't recognized by the Notes client. So, when you run the application in the

Notes client, the pass-thru HTML won't function. In R5, if you move that pass-thru HTML to the appropriate place in the IDE, the Notes client will recognize it.

So, while you can leave your applications as they are when you upgrade, it's worth your while to migrate your code. When your JavaScript code is handled in the programmer's pane, it is syntax-checked and color-coded. You'll also be able to find-and-replace within a script, across all scripts for a given object, or across all scripts for all objects on a form. You won't have this capability if your JavaScript code is hosted as pass-thru HTML.

What is the JS Header placeholder and how does it help users?
This is really a Domino form "event" that is the place in the IDE where you can place JavaScript functions for your applications. The information in the JS Header is exactly what Web developers put within a <SCRIPT> tag hosted inside the <HEAD> area of a document. It's a good place to put function definitions or global declarations that you'll need in other places on the form.

Can you give me an example of something that JavaScript can do that LotusScript can't do?
Well, run on the Web for one. But, that aside, you can use the combination of JavaScript and the DOM to create interfaces on-the-fly for the client. For example, suppose you're exposing a "query builder" interface to a user. When the user wants to add another "clause" to the query, the JavaScript/DOM connection can actually create whole new fields on-the-fly. If you use Internet Explorer 4.0 or higher and you would like to see an example of this, see the sidebar, "**Query Builder Example**."

Is this the end of LotusScript?
No way! Some developers may choose to use JavaScript to create applications that run on the Web. Some developers may choose to use LotusScript because they are familiar with the language and appreciate its capabilities. LotusScript is still a great Notes client scripting language, and we will continue to support and extend it. It is particularly handy for accessing OLE objects, or making a transition from Visual Basic.

Plus, independent of the client type you want to target, LotusScript is also a great server-side scripting language. The beauty of our architecture is that when we add new objects to the back-end classes, all of our language bindings inherit that functionality.

What is "LiveConnect"?
LiveConnect is what allows JavaScript to access components. Just as you can use an "automation" capability to manipulate an OCX written in C++ using Visual Basic, you can use the "LiveConnect" capability to manipulate an applet written in Java using JavaScript. If you know the API for a component, you can drive the component using JavaScript without any concern as to what language the component itself was written.

For example, if there is a calendar applet on a page, then JavaScript could set the calendar to show the "current" date, or bring up the month of January, 2000, or change to a weekly view, and so on. In essence, JavaScript has access to the entire public API of the component. Another way of looking at this is that when you lay an applet down on a page, you've automatically extended the DOM!

The LiveConnect capability is part of the three major clients: Netscape Navigator, Internet Explorer, and now, Notes R5.

Can you talk about Common Object Request Broker Architecture (CORBA) in relation to JavaScript? Or how you can access a JavaScript application's back-end classes?
Okay, I like the second question better. This just happens to be one of the

coolest things about our JavaScript support in R5.

In R4.6, we started building a Java interface to the back-end classes. With R5, we've enhanced the back-end classes, given them a full Java API, and created an applet (called lotus.domino.AppletBase) that allows you to access the back-end from Java. So, combining this applet with the miracle of LiveConnect and a little bit of JavaScript, you get client-side access to the back-end objects. LotusScript programmers do this all the time; they constantly traipse around the back-end with little concern for the fact that they're running client-side scripts. Well now you can do the same thing with JavaScript!

But let's address your CORBA question. It just so happens that when you're running the AppletBase applet in the Notes client, we use the native Notes RPC protocol to access the back-end. When you're running in a browser, we use the IIOP protocol to talk to a server-side ORB to access the back-end. This just happens for you automagically; Domino knows which protocol to use in which case.

Let me reinforce this statement. The AppletBase applet is not a CORBA applet. It simply provides easy access to the back-end objects. Sometimes it will use CORBA's transport protocol; sometimes it will use another protocol. Beyond R5, we're set up to augment the set of protocols we use, but Notes developers don't have to concern themselves with this. The API to the Domino Objects remains constant; the protocol can vary behind the scenes to take advantage of a given platform or to tune the performance.

What was your role in the JavaScript effort?
I was the thorn. I kept going around pontificating: "Who needs LotusScript? Who needs the formula language? They don't work on the Web. Give me JavaScript!" People got pretty tired of me. But the point was to put on the blinders, to be the advocate for the Web developer, and to help define a JavaScript experience for Notes developers in general. I adopted the moniker of "JavaScript Bigot", which is not only far less pompous than "JavaScript Evangelist," but it also freely acknowledges the "blinder" effect.

A tremendous amount of work went into the JavaScript effort. The Languages team had to host the JavaScript interpreter into the Designer and client, and then hook it up through LiveConnect to our Java Virtual Machine. The Designer team had to incorporate JavaScript into the IDE and create the hooks for the W3C HTML 4.0 events. The Domino team had to serve up these events faithfully and generate further JavaScript wiring to make it all hang together properly -- not an easy task given the various flavors of browsers out there. The Notes client team, with the biggest challenge of all, had to map the DOM onto the existing Notes editor. The Applets group had to develop the Java components that JavaScript wires together to create a Notes experience within a browser. The Notes Back-end team had to re-architect the Domino Objects to make them accessible using Java and JavaScript from both the Notes client and a browser client. The template developers had to tweak their approach to map onto a more Web-centric paradigm. And then there's the QE effort, the documentation effort, the HelpDesk effort, and so on...

So you can see that I had the easy job. I just sat in the middle and kept nudging everyone so that it would all come together.

What are we doing beyond R5 with JavaScript?
The first order of business is to improve the level of DOM support in the Notes client to make it HTML 4.0 compliant (and beyond). Both the Designer and the Notes client will need to support CSS to do this. Better JavaScript library support will be added to the Designer as well as support for in-line JavaScript. We'll also be creating more Java applets and JavaScript libraries that mimic Notes functionality.

We expect that the DOM (as defined by the W3C standardization effort) will

become the way that users manipulate client-side documents. We'll be melding the NotesUI classes into the DOM model. The Notes UIView, for instance, will become an object/applet that supports an API that is the union of the current NotesUIView and the ViewApplet. You will access a NotesUIView in the DOM as if it were an <OBJECT> tag on an HTML page.

More and more of our internal components will be exposed externally. Customers will be able to subclass our components to better customize their own applications. If you don't like our Time Picker, for example, you'll be able to plug in your own. The JavaScript access to the customized component remains invariant.

In addition, XML interfaces have only been used internally in R5, but will become externally available. The Notes client will come to support an XML DOM naturally accessed using JavaScript. And, most important to me, because I'm a LotusScript and formula language neophyte, I'd like to see JavaScript to appear in the server-side scripting engine!

Any last words?
Well, here's a final prognostication: DHTML is not just for the Web anymore. From my perspective, it's becoming the windowing layer for all applications on all platforms. Whether it be Windows or UNIX, GUIs will be written in DHTML soon, not C++. And remember, JavaScript goes hand-in-hand with DHTML. (But then, I'm only the "JavaScript Bigot".)

BIOGRAPHY
Mark Judd is the "JavaScript Bigot" here at Iris. In previous lives, he was a Java bigot (actually, he still is), and an Ada bigot (no longer). At Iris for 2 years now, he first worked in the Java Applet group before concentrating on the JavaScript and Dynamic HTML effort. Prior to Iris, he built software for integrated digital services over a fiber optic network (20 years ago), videotex systems (15 years ago), Ada IDEs (10 years ago), 4GL IDEs (5 years ago), and (more recently) Java/Web applications.

Outside of work, Mark and his wife Terri both find themselves being soccer moms to their kids Zak, Zoë, and Alexis. Though he's given up on a White Christmas, Mark's still hoping for snow so he can flood the backyard and get the kids out skating (or maybe curling).

[back to "Mark Judd: JavaScript integration"]

Query Builder Example (sidebar)

This example requires Internet Explorer 4.0 or higher. To use this example, click on the bottom link to create rows dynamically. Click on the "X" buttons to delete a row. You can also try building a query that includes a clause that limits the search to documents created between certain dates.

```
[<script>
function fieldChange (obj) {
  var rowType = obj.value;
  var container = obj.parentElement.parentElement.all;
  container.comparison.outerHTML =
    genSelect("comparison", rowType.indexOf("Date") == 0 ? dateOpts : textOpts, null,
"comparisonChange(this)");
  container.sfield2.style.display = "none";
}
function comparisonChange (obj) {
  var container = obj.parentElement.parentElement.all;
  container.sfield2.style.display = (obj.value.indexOf("between") == -1) ? "none" : "";
}
function genSelect (selectName, opts, chosenOption, doChange) {
  var change = doChange ? (" onChange='" + doChange + "'") : "";
  var name = " NAME='" + selectName + "'";
  var s = "<SELECT" + name + change + ">";
  for (var i = 0; i < opts.length; i++) {
    var opt = opts[i];
    var value = " VALUE='" + opt + "'";
    s += "<OPTION" + value + ">" + opt + "</option>";
  }
  s += "</SELECT>";
  return s;
}

var dateOpts  = new Array ("is on", "is after", "is before", "is between", "is not between", "is not on");
var textOpts  = new Array ("contains", "does not contain");
var conjOpts  = new Array ("and", "or", "not");
var fieldOpts = new Array ("Author", "Date Created", "Date Modified", "Text", "Title");

function deleteRow (obj) {
  document.all.searchTable.deleteRow(obj.rowIndex);
  document.all.searchTable.rows[0].all.conjunction.style.visibility = "hidden";
}
function addRow () {
  var newRow = document.all.searchTable.insertRow(document.all.searchTable.rows.length-1);
  (newRow.insertCell()).innerHTML =
    "<INPUT TYPE=BUTTON value='X' id='deleteMe' onClick='deleteRow(this.parentElement.parentElement)'>";
  (newRow.insertCell()).innerHTML = genSelect("conjunction", conjOpts);
  (newRow.insertCell()).innerHTML = genSelect("field", fieldOpts, null, "fieldChange(this)");
  (newRow.insertCell()).innerHTML = genSelect("comparison", textOpts, null, "comparisonChange(this)");
  (newRow.insertCell()).innerHTML =
    "<INPUT TYPE=TEXT id=field1><SPAN id=sfield2 style='display:none'> and <INPUT TYPE=TEXT
id=field2></SPAN>";
```

```
   document.all.searchTable.rows[0].all.conjunction.style.visibility = "hidden";
   document.all.searchTable.rows[0].all.deleteMe.style.visibility = "hidden";
}
</script>
<TABLE id=searchTable>
<TBODY VALIGN=top>
<TR><TD></TD><TD></TD><TD colspan=3><A HREF=# onClick="addRow()">Click to add a new search
clause</A></TD></TR>
</TBODY>
</TABLE>
<script>
document.body.onload=addRow;
</script>]
```