Domino and JavaScript: Dynamic Partners (part I)

by David MacPhee, karen Hobert, and Russell Lipton

[Editor's note: This article resides in "Notes Today", the technical Webzine located on the http://www.notes.net Web site produced by Iris Associates, the developers of Domino/Notes.]

Introduction

Notes is extraordinarily well-suited to building dynamic JavaScript applications that control the user environment on the Web, taking advantage of forms and views to store, manipulate and publish scripts conditionally to the browser.

This article introduces a series of articles focusing on the use and integration of Domino and JavaScript. Future articles will include an in-depth look at the implementation of Domino.Merchant and recipes for including key JavaScript functions in your Notes applications to manage client-side processing of Web applications.

Java: The Short Course

A brief introduction to Java will help us place both it and JavaScript into the appropriate context. Sun Microsystems originally developed the Java specification for support of small consumer devices. When the Internet became prominent, they realized that Java would be ideal for developing reasonably secure and highly distributed network applications across the Internet.

As a language, Java combines many of the best features of C++ and Smalltalk. Like C++, it supports strong type-checking and compilation. Like Smalltalk, it is strictly object-oriented and eschews pointers for automated garbage collection of the memory space.

Unlike either C++ or Smalltalk, Java was designed with an onboard security model. By default, Java applications do not gain access to underlying file systems. This is a double-edged sword, since powerful enterprise applications must, of course, manipulate these systems. The use of large Java applications will probably proceed in tight coupling with the evolution of accepted security solutions.

Java also provides sophisticated support for multi-threaded programming. Long-term, this will probably prove to be Java's key differentiation.

While Java seems to be somewhat easier to learn than C++, taking advantage of its advanced features is challenging. Also, because its distributed design emphasizes platform-independence, there has been some grumbling over the clumsiness of Java's AWT framework for building graphical interfaces.

Because Java is a pure object-oriented language that relies on its class hierarchy at run-time, all classes needed to execute even small applets must either be found resident on the client or be downloaded to the client before an application launches. This imposes performance penalties unlikely to disappear in the near future, both because of general bandwidth limitations as well as the continual development of more functional class libraries that any given client is unlikely to possess.

Despite these rather significant nits, both interest in as well as use of the language has exploded since its introduction in 1995. Java today is a classic case of the half-full glass. Considering its newness, it is remarkably mature. Considering the complexities of supporting enterprise-wide, distributed applications, it must still be viewed as immature.

Where Java has already proven its value is in the creation of small, portable applications (applets) that can be dropped into development or user environments. These applets can expose selected properties (for instance, size, color, position et al) for manipulation by the client at run-time.

While we emphasize JavaScript in this series, we used Java to great advantage in our work. For instance, we created a Java applet that enables Domino.Merchant users to describe how a page is to be laid out for rendering by

the module that assembles the user's site. Essentially, the Java applet takes the description and builds a simulated, viewable model of the desired page. Once the user has signed-off, the choices are put back into a Notes document and passed down to Domino.

JavaScript: Opening Up The Browser

The best way to understand JavaScript is that JavaScript is to the Web browser model as LotusScript is to Notes. Just as LotusScript exposes Notes classes to manipulation for application development, JavaScript exposes the browser object model.

The term "browser object model" here refers to the collection of objects that are managed by the browser. These objects include such indispensable items as windows, frames, documents and forms. Browser products from Netscape and Microsoft consist essentially of a user interface wrapped around these objects. Microsoft supplies an Active/X control that separates the Internet Explorer UI from the objects. An earlier Notes.Net article described how to integrate this control into Notes applications. Domino 4.6 will integrate this capability directly within the environment.

To avoid the learning curve, compilation and performance costs of Java, JavaScript largely mimics an object-based VisualBasic-style language design, though without an onboard integrated development environment. Netscape has promised delivery of an IDE for the near future.

While JavaScript cannot define new classes of objects, it can build new object instances from the objects already defined by the browser (for instance, a new window object instance or document object instance). Associative arrays enable arbitrary data values to be associated with arbitrary strings, giving the ability to define new objects beyond those already supplied.

Although JavaScript was not designed for security, it, like Java, prohibits access to a user's underlying file system. JavaScript binds its objects at run-time and is interpreted.

Where Java applets exist as compiled executables that are discrete from (though called by) HTML code tags, JavaScript is directly described within HTML documents. This is by design, not by accident. Scripts can interact dynamically against user input at run-time, even to the extent of assembling JavaScripts themselves in response to this input for subsequent execution.

The JavaScript execution model works as follows:

- You define JavaScript code within a HTML document in the form of object instances. Functions, methods and properties are bound to these object instances as needed. Again, this code coexists within an HTML document along with "classical" HTML tags (see example).
- Then, when a document is loaded by the browser, the browser parses any JavaScript code and caches it.
- Finally, appropriate HTML tags trigger execution of a JavaScript function on an event-driven basis (for example, as the result of a "click").

Recently, Netscape has added LiveConnect technology to the mix. JavaScript can now gain access directly to Java variables, methods, classes and packages that have been exposed explicitly by a particular applet. This includes the ability for JavaScript to selectively modify Java properties at run-time. Java code can, in turn, access JavaScript methods and properties.

As a neat corollary, plug-ins can define a Java class structure that exposes some of their properties to JavaScript as well, giving JavaScript the ability to drive plug-ins intelligently at run-time.

Though JavaScript's features are nifty, often seriously nifty, the bottom-line benefit is the enhanced ability for making Web applications more interactive on the client side. Every time a Notes application user can execute

behavior on the client, rather than take a hit (almost always, multiple hits) against the Domino server, performance gains are realized.

As Notes applications are used increasingly by a mixed Notes and non-Notes user population on the Web (indeed, as it becomes impossible to predetermine the expected proportion of these populations), the ability to use JavaScript appropriately may mean the difference between a successful and an unsuccessful Domino application.

The Architecture of Domino.Applications

A few words about the architecture of Domino.Applications will help us understand how we were led to a new model for developing Notes applications when we implemented Domino.Merchant. Each Domino.Application is layered around Notes and inherits the rich set of services common to all Notes applications (object store, security, replication and distribution). Currently, Lotus is shipping or preparing to ship this collection:

- Domino.Action (included free with Domino)
- Domino.Broadcast (utilizes Pointcast servers for pushing information to browsers)
- Domino.Doc (a document management and versioning application)
- Domino.Connect (back-end access to relational and transactional systems)
- Domino.Merchant (tools for developing Internet commerce sites, including secure ordering)

For simplicity, we will restrict ourselves here to Domino.Action and Domino.Merchant. These applications can be conceptualized as a set of wizards and templates that use a common control module to specify, build and generate a web site.

A web site is constructed as a group of discrete areas. A "home page" is an area. So is a discussion group or a product catalog. Each area generates its own .nsf database. All design elements, many of which are supplied pre-built by Lotus (for instance, backgrounds and other "looks"), are retrievable from a shared library. Thus, Domino.Merchant contributes a set of areas germane to commerce (storefronts, secure ordering, etc) for integration into the library.

Edits and updates are posted to a common control module, the "SiteCreator". SiteCreator analyzes the data collected from the user during specification and invokes needed APIs that call an application assembler. This AppAssembler reads the Merchant documents and builds the required area databases. As far as possible, defaults are collected once and then mapped across the generated site.

Of course, Web sites change frequently. Future changes to existing templates can be made offline, using the same wizard-driven process, and SiteCreator can be re-invoked to regenerate the site.

Notes programmers can also define entirely new area types or page types within an area through a mixed process of cloning and modification of existing areas. Since specification documents are stored in the library database as a typed group of subforms (for example, as a commerce storefront in Domino.Merchant), this document can be inspected, edited and then pasted in place of the corresponding specification document within SiteCreator.

Because Domino.Merchant is a serial, process-driven, Web-centric application that collects data from users, we needed to take control of the client session. This led us to JavaScript. What started as an application necessity evolved into a series of exciting discoveries about the synergy between Notes and JavaScript that will be the subject of future articles in this series.

Web-Think: A New Application Model

Life is too short to add new technologies for their own sake. Our comparison of JavaScript and LotusScript naturally leads to the question, "if the two are so similar, why not just use LotusScript and Domino's ability to convert Notes to HTML and punt on learning still another programming language?"

The short answer is that you can do this and that it will become easier as time goes on. We don't have to be industry pundits to assume that Notes will ultimately provide internal representation of its data in HTML form. Added to this, Fusion, the superb site development and management tool which IBM has acquired, with future integration slated for Domino, generates HTML.

However, while Domino developers can certainly deploy interactive Web applications relying heavily on conversion of existing (or newly created) Notes-centric designs, a Web model is emerging for user interface, search, navigation and application interaction that overlaps but does not converge entirely with Notes.

Web user interfaces and Notes user interfaces are still more disjunct. As network computers and WebTV devices emerge, we can expect JavaScript to offer the most direct support for programming suitable styles and interfaces for these devices.

Also, recall that JavaScript was developed not least to facilitate client-centric computation and interaction that was burdening (and still burdens) Web servers. Domino is no worse but no better than other Web servers in this respect. Judicious use of JavaScript will enhance both the actual and perceived performance of Web applications. LotusScript can't help you there.

The ability to generate JavaScript dynamically from Notes speaks to the opportunity for putting serious, interactive applications onto the Web. This is far more than publishing static pages from a web site, though, of course, it includes that. Of course, Domino can already do this for the server. JavaScript nicely enhances its efficiency and power to do this on the client side.

Guidelines For Choosing A JavaScript Strategy

Future articles in this series will describe the implementation of Domino.Merchant and share some of the more important JavaScripts that make it an unusual Notes application. We offer here some guidelines for deciding when JavaScripts are likely to be useful in your development.

You should strongly consider the use of JavaScript in these situations:

- *When your application will be hosted for primary access by Web browsers, and especially when many of the users are unfamiliar with a Notes user interface (twisties, views and the like).* We built a folder-document UI widget in JavaScript to reduce the cognitive dissonance for this audience.
- *When your application needs to take over the user interface to drive a step-driven interaction that collects data for subsequent iterations with the user.* This applies to wizard-like sequences and may be relevant for certain aspects of an application, even though other aspects remain more classically Notes-driven (collaboration, mail, discussion groups). A better way to put this is that JavaScript enables you to retain the best aspects of Notes applications while enabling you to build new types of applications with richer behaviors.
- *When you want to reduce network chattiness between browser-client and the Web server.* This is equivalent to saying that you should use JavaScript whenever it supports the same interaction within the client that can be supported between the client and the server, since it is always beneficial to reduce traffic with the server. JavaScripts can be downloaded once and cached for use throughout the client session.
- *To simulate Notes functional features on a Web browser (for instance, dialog and info boxes) that will provide a consistent look-and-feel for mixed-use client environments.*
- *To validate data at the field level, rather than at the server level.* This is not only a performance but a functional issue. Well-written JavaScripts can provide extremely friendly context-sensitive help, and enable the cursor to be placed in the offending field on the display.

The above is not an exhaustive list. It tends to underestimate the benefits you will gain as mastery of JavaScript helps you envision entirely new types of Notes applications.

There are numerous resources on the Web that can help you learn JavaScript. One downloadable tutorial you may find helpful is the one authored by Netscape itself :
http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/index.html.

Still another great introduction to JavaScript, written by Stefan Koch, is downloadable from the Net as well: http://rummelplatz.uni-mannheim.de/~skoch/js/tutorial.htm.

Once you feel you're ready to start examining some working scripts, we suggest taking a look at some of Gordon McComb's JavaWorld columns. You can get an index of the columns from his own site:
http://gmccomb.com/javascript/index1.html. Too often, JavaScript examples are presented as toys. McComb's sample scripts are useful.

Java, JavaScript, LotusScript: When and Where?

What happens to your hard-won knowledge of LotusScript in this new Internet-driven world? Where do Java, JavaScript and LotusScript fit together, if indeed they do fit together?

LotusScript will continue to be your tool of choice for manipulating the Notes application model, especially for applications that are specialized for Notes networks. However, as Domino internals are increasingly matched for support of Internet data formats and protocols, we can reasonably speculate that LotusScript will offer increased ability for sophisticated conversion and mapping of Notes applications to the Web. Still, LotusScript will remain primarily a tool for leveraging Notes itself.

JavaScript exposes the browser model much the same way that LotusScript exposes the Notes application model. As browser capabilities become richer, JavaScript will evolve to offer highly granular support for manipulating client interactivity. While JavaScript is actually a proprietary product for Netscape, Microsoft and others are developing reasonably compatible versions. At the current time, JavaScript offers Domino programmers the best means for developing applications whose primary target are Web users, especially if many of those users are unfamiliar with the Notes user interface.

Java is a full-blown programming language that can support stand-alone as well as small, network-distributed applications (applets). Most likely, Java will largely replace C++ over the coming years as the product of choice for building highly modular, object-oriented systems, servers, development tools and application frameworks. In the short term, unless you are a systems programmer, you probably don't need to make the significant investment to become a Java programmer yourself.

Summing up, your time-tested LotusScript skills remain entirely relevant to your Domino development into the foreseeable future. We do recommend that you master JavaScript, because it will offer you exciting opportunities to develop Web-centric applications that meet the current requirements of both Notes and non-Notes users. Meanwhile, keep a weather eye on Java and take advantage of interesting Java applets as they become available.

Next: Domino.Merchant Implementation and JavaScript Components

The next article in this series will describe the implementation of Domino.Merchant in detail. We will emphasize how we used existing Notes features to gain the maximum possible leverage and the way that JavaScripts are passed as arrays from Domino to the client itself.

Of course, we found ourselves implementing specific JavaScript components of lasting value not only to Domino.Merchant but to your own development as well. We will share these in future articles in this series. They include:

- **Field validation**
  We are not the first ones to use JavaScript for validating fields within client browser forms. In fact, this probably remains to this day the most widely used aspect of JavaScript functionality. But we will show you scripts that use a single Notes field to generate code for tracking Notes as well as JavaScript clients for

common browser rendering. Not only do these scripts perform validation, but they also display error messages as appropriate with context-driven display within particular fields. All messages are managed centrally to enable powerful language localization.

- **Address book lookups**
  It is hard to improve on the Notes address book implementation, so we elected to simulate it as needed within Web browser-driven applications. These JavaScripts present a simulated Notes address book, enable users to select groups and persons for managing ACLs from the browser. The UI presents a dialog box effect (actually, we spawn a window, to use JavaScript terminology). Graphics (.gifs) are used to enhance the dialog simulation effect.

- **Browser frame manipulation**
  In many ways, frame manipulation is where JavaScript shines, but it also demands the most knowledge both of JavaScript and the browser model itself. Knowing when to download scripts to frames and which frames to target for script execution is crucial.

Stay tuned!

ABOUT DAVID AND KAREN

David MacPhee and Karen Hobert of BadDogBad are long-time Notes developers who have worked closely with Lotus to design and build Web interfaces for Domino.Applications.