Domino Designer 7: JavaScript

Version 1.0

Copyright Information

Part #DDJS7-1.0, updated for Domino Designer 7.0.1.

Table of Contents

Description

During this course you will use Domino Designer 7 to add JavaScript to Domino applications that are accessed by browsers. The course covers the basic language elements of JavaScript, how to add scripts using Domino Designer, and how to exploit the various browser and language object event handlers. There is a strong emphasis on the browser object model and how it relates to the Domino object model. The course also touches on how to incorporate Dynamic HTML, Java Applets, LiveConnect, ActiveX controls, and AJAX into web-based applications.

Course goals

This course will:

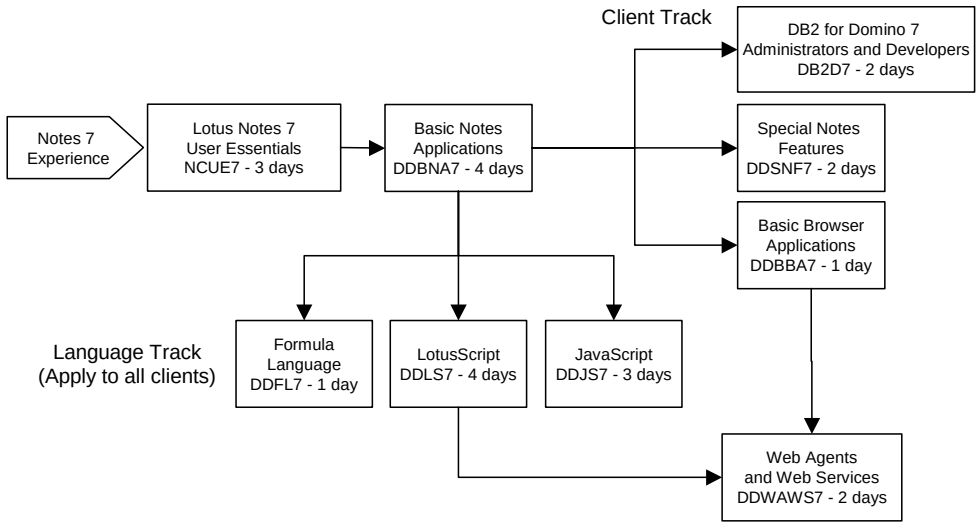
- build a fundamental knowledge of JavaScript as it is applied in Domino applications used by modern browsers
- provide practical programming and debugging experience to ensure a foundation of JavaScript skills
- understand the relationship between JavaScript and Domino data types
- clarify the use of the various object models, including the Domino Object Model, the original browser Document Object Model, JavaScript language objects, and the newer W3C Level 1 Document Object Model
- use JavaScript to code the Field, Button, and Form event handlers
- build a practical understanding of data validation and error trapping
- use DHTML to create interactive web pages
- control Java applets using LiveConnect
- remotely access Domino objects via CORBA
- script ActiveX objects
- use AJAX to request and process Domino-generated XML
- develop practical ways to detect which browser is being used and how to code appropriately.

Audience

This course assumes that you have:

- thorough knowledge of the Domino Designer 7 development environment, including Form, Page, View, Frameset, and Agent design, as well as how to set properties and set the ACL
- knowledge of Web technologies, including servers, browsers, HTML, Cascading Style Sheets, and some basic JavaScript (or other browser scripting language) and awareness of browser object properties and methods
- basic understanding of LotusScript and the various Notes product objects
- because this course does *not* review any aspects of the non-JavaScript aspects of developing applications with Domino Designer, mastery of the topics covered in these courses:
  - *Lotus Domino Designer 7: Basic Notes Applications*
  - *Lotus Domino Designer 7: Basic Browser Applications courses.*

This course is part of a series of Domino Designer 7 training courses. Follow these paths to master all aspects of developing applications using Domino Designer:



Domino Designer 7: Basic Notes Applications provides the base knowledge for this additional training:

Client Track

- Configure Domino to use DB2 as a database engine and build applications that access relational data, *DB2 for Domino 7 Administrators and Developers*.
- Specialize in programming Notes applications to respond to built-in user interface features, *Domino Designer 7: Special Notes Features.*
- Convert an application written for Notes clients to be used in browsers, *Domino Designer 7: Basic Browser Applications.*
- Provide data to cutting-edge Web applications, *Domino Designer 7: Web Agents and Web Services*.

Language Track. These languages apply to both Notes and browser clients:

- Learn sophisticated application techniques that fully exploit the formula language in *Domino Designer 7: Formula Language*.
- Develop sophisticated applications and Agents with LotusScript in *Domino Designer 7: LotusScript*.
- Add powerful client-side scripting to browser applications in *Domino Designer 7: JavaScript*.

Course design

The course takes a task-oriented approach, during which you will work with JavaScript code that will have immediate application to your Domino applications.

Because this course instructs you how JavaScript works with respect to Domino, you will be able to leverage the many JavaScript resources available to you as a developer.

Data files

To ready your computer and the Domino Server for the course, you must run the **`INSTALL.EXE`** program from the class diskette on the computer running Domino Designer. Specify the default Notes \Data directory during the installation; if you put it into a subdirectory outside the \Data directory, the examples will fail. By default, a subdirectory named \Data\DDJS7 is created for you.

The *DDJS7 Demo* (`DDJS7DEMO.NSF`) database is installed. It is used to demonstrate all the code used in the course and as the starting point for the exercises. During the first exercise you will create your own non-replica copy of the *DDJS7 Demo* database on the Domino Server, where you will complete the exercises.

There are also a number of other resource files that will be installed to your local drive (sample ActiveX controls and Java Applets) that you will use during the course.

All exercises must be performed from your copy of the database running on the Domino Server and tested from there. You should have Domino Designer 7 and Internet Explorer 6+ installed on your computer (IE will run all the example code). You should also install Mozilla Firefox 1.5+ to test cross-browser compatibility.

Be sure you have access to *Domino Designer 7 Help*, which should be full text indexed.

Conventions

This course follows these font conventions:

- *Italic* - database, view, form, document, macro, and field names, as well as object event handlers
- **Bold** - menu options, command button names, and accelerator keys, function/statement names (for clarity)
- `Courier` - user input, sample values, code examples, constants
- **Helvetica** - HTML and JavaScript code examples
- ↵ - shows when script lines wrap in the text but should be one continuous line in the Programming Pane.

Notes

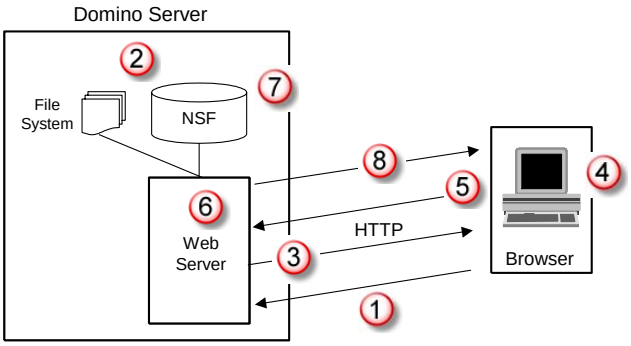
Topic 1: Browser Objects and JavaScript

Key points

This Topic provides a historical and conceptual background of how JavaScript relates to and interacts with browsers. It also describes how Domino Server dynamically and securely serves pages containing JavaScript to browsers.

Preview of the Domino/browser exchange

This diagram illustrates the exchange between Domino and a browser (you will learn more about all the pieces in the following sections):



This table describes the exchange between Domino and a browser:

Stage	Description
1	A browser requests a URL that is on the Domino server. In this example, the URL opens a Form in which the user will add values to several fields and will save the results back as a document in the NSF on the server. Pretty simple stuff.
2	Domino pulls the necessary resources from the NSF, in this example, a Form.

Stage	Description

3	The Form itself is designed in Domino Designer and Domino is responsible for converting various design elements into HTML:
	• Fields are converted to HTML **<INPUT>** tags.
	• The Save button that is designed to run **`@Command([FileSave]); @Command([FileCloseWindow])`** is converted to JavaScript that submits the document back to the server when clicked.
	• The Form may also have text that has embedded HTML tags (including inline **<SCRIPT>** tags) and has been marked as Passthru HTML, so Domino doesn't need to do any conversion.
	• The Form may have "passthru" elements, where code was added in Domino Designer in various browser-specific places, such as:
	  • **HTML Head Content**. Added to the HTML page **<HEAD>** tag (uses the Function language to dynamically define content).
	  • **JS Header**. JavaScript added to a **<SCRIPT>** block in the page **<HEAD>**.
	  • **OnLoad**. JavaScript added to the **<BODY>** tag onLoad event handler.
	• If the Form calls an Agent in its WebQueryOpen event, the code (written in LotusScript or Java) is executed (often used to programmatically set default field values).
	After all the computations are performed (regardless of the original incarnation of any element), the Form is converted entirely to browser-compatible HTML and sent down to the browser via HTTP.

Stage	Description
4	The browser renders the elements of the HTML page. In this example, the Form has an **<IMG>** tag pointing to a .JPG in the \HTML directory on the Domino server, so as the browser renders the page, it fetches the image. You can think of the page as having a visual aspect, what the user sees, and code. Some of the code was run before the visual pieces were rendered (like the code in the onLoad event), but other client-side code isn't executed until the user takes some action, like clicking a button. The point to be made here is that all "stuff" of the visual aspects and client-side code buried in the HTML page has been shipped down to the browser from Domino, and it is now up to the browser to deal with it. What this means for the developer is that you must deeply understand at least these two things: • How Domino converts (or passes through) things you add in Domino Designer. • What the browser does with page once it arrives (including whether or not the browser is capable of processing the page).
5	In this example, the user clicks the **Save** button and the page is submitted back to Domino. With respect to the **Save** button, as the Form developer you not only had to add all the things necessary for the page to render and operate properly in the browser, you also had to anticipate what happens when the user is done with the page. In this case you added an image of a **Save** button that when clicked submits the Form back to the server. A Form may also have a **Cancel** button, or a **Go Back to the View** button; a View may have a **Next Page** button to show the next 20 documents in the View, or a **New Document** button. So now because you also have to anticipate user navigation away from the page, you now have three things about the Form you must worry about.

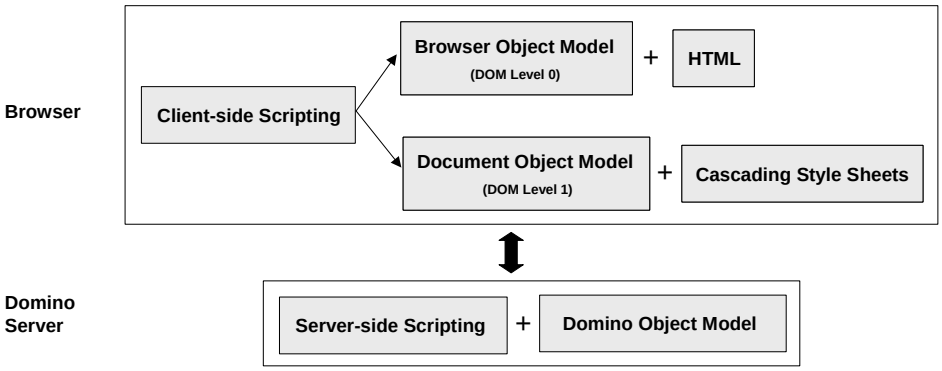
Stage	Description
6	Domino receives the page from the browser and is now responsible for transforming its information into a document in the database. To do this, Domino fetches the original Form to interpret the field data. It runs the field formulas (computed, input translation, input validation). If there are input validation failures, Domino sends an error page back to the browser. If the Form calls an Agent in its WebQuerySave event, the code (written in LotusScript or Java) is executed (often used to more gracefully handle field validation errors or to redirect the user to another place in the application after successfully writing the document to the database).
7	Once all the conditions and code of the original Form have been met, the document is created in the database.
8	Depending on what you code to happen next, Domino can return something back to the browser, like an error or confirmation message, back to the Form for error correction, or to the next logical place in the application. This is another opportunity for your code work, but also another realm of worry.

Programming challenge

Programming Domino applications for browsers is really quite a challenge. As you saw above, there are several opportunities to apply code:

- Server-side code that runs before the HTML page is shipped down to the browser.
- Client-side code that runs entirely in the browser and eventually is responsible for posting something (usually the HTML form) back to the server.
- Server-side code that processes the response from the browser.

In this course, the emphasis is on the client-side code, specifically how JavaScript can be used to manipulate browser and Document Object Model (DOM) objects created by tags included in the current HTML page and by JavaScript language itself:



This course also necessarily must also deal somewhat with Domino as the Web Server, so you also will work with the Domino Object Model and how to use:

- @Functions to create Views as well as to code buttons and Action buttons using a subset of @Functions (the code is automatically converted to JavaScript by Domino).
- LotusScript (and/or Java) for powerful server-side processing such as to dynamically create HTML pages or perform post-submit processing.

The history of client-side scripting

In the beginning, the idea of a Web browser was to display crudely formatted text and images using a simple markup language (HTML) and provide hyperlinks to other pages. It only took a few years for this simple display-only approach to change drastically to where the browser is now a universal user interface for custom applications.

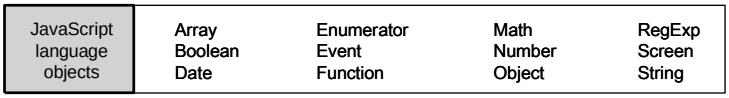
Scripting in browsers is implemented as a runtime interpreted language that cannot run outside the context of an HTML page interpreted by a browser. This is because the code is actually sent in human-readable text along with the HTML content to the browser for local execution. The browser renders the HTML it finds and executes the script from various events.

JavaScript has its own development history parallel to but separate from browsers. It should really be thought of as independent from the browser object model because there are browsers that support other scripting languages (e.g. Perl, Java, Python, etc.) and because JavaScript is also used in non-browser products (e.g. Windows Script Host, Groove, etc.).

For simplicity, this course focuses on “JavaScript” as a generic label for the various scripting language variants used in the most popular browsers:

- JavaScript is the scripting language used to script browsers with a Netscape lineage, such as the new Mozilla/Gecko-based browsers.
- Microsoft Internet Explorer uses JScript (as well as VBScript), a JavaScript variant that is COM-aware.
- ECMAScript (the European Computer Manufacturers Association's ECMA-262 language) comes from the Web Standards Project, and is an attempt to standardize the native language elements and language-based objects of browser scripting. Both Microsoft and Netscape claim to be ECMAScript compatible, though each adds extensions that the other cannot interpret. Mozilla, for example, follows as much as possible the ECMAScript standard. The Notes client also uses a JavaScript interpreter that follows the ECMA standard to a degree (ECMAScript 1.4, though it lacks a complete browser DOM, as you will see below).

JavaScript is a somewhat full-featured scripting language with variables of several data types, various math, string, equivalence, and Boolean operators, statements, functions, arrays, error trapping, objects, and object events/methods/properties, giving it the ability to perform computations and interact with users via dialog boxes. The JavaScript language objects are used to manipulate data, which may/may not be derived from browser objects:



Brief history of browser object models

Using JavaScript to code browser behaviors has always been challenging because of the rapid development cycle of browsers. With each browser brand (Internet Explorer, Netscape Navigator, etc.) and version comes a new and improved object model. Because you generally can't control which browser brand or version is installed on users' computers, your code has to either find a lowest common denominator or branch to accommodate the many possible browser brand/version combinations. You’ll learn more about browser compatibility late in this course.

When JavaScript was first introduced to browsers, the first object model was the *Browser Object Model,* now known as the Document Object Model (DOM) Level 0. The early browsers Netscape Navigator 2 and Internet Explorer 3 supported the Level 0 DOM.

As browsers developed, Microsoft and Netscape developed competing and incompatible proprietary DOMs in their Version 4 browsers to support Dynamic HTML and Cascading Style Sheets. It would be convenient (but alas impossible) to ignore these models because the differences are both significant and aggravating. While IE continues to support most of its first DOM in its current version, Netscape completely abandoned its version 4 DOM and has replaced it starting with Netscape 6 with the W3C DOM. When Netscape development transferred to the open source Mozilla Organization in 1998, the Mozilla browser continued with implementing the W3C DOM.

These are the standards-based, vendor-neutral models promoted by the W3C (**http://www.w3.org/DOM/**, summarized nicely at **http://www.mozilla.org/docs/dom/reference/levels.html**):

- DOM Level 1 incorporates and extends Level 0 as well as provides full support for *Dynamic HTML* (DHTML). Level 1 is supported to some extent by most current "5th generation" browsers such as Netscape 6+ and Internet Explorer 5+ (as well a bevy of standards-based browsers such as Mozilla 1+ and Opera 7+). This is considered the current DOM.

- In 2000, DOM Level 2 added a style sheet object model for easier access to styles attached to a document. It also introduced an event model and support for XML namespaces. The 5th generation browsers (e.g. Mozilla) support some aspects of DOM Level 2, but not all.

- In February 2003, the W3C published the 1.0 specification for DOM Level 3. (**http://www.w3.org/TR/DOM-Level-3-Core**). It extends DOM Level 2 primarily to enhance support for XML/XHTML.

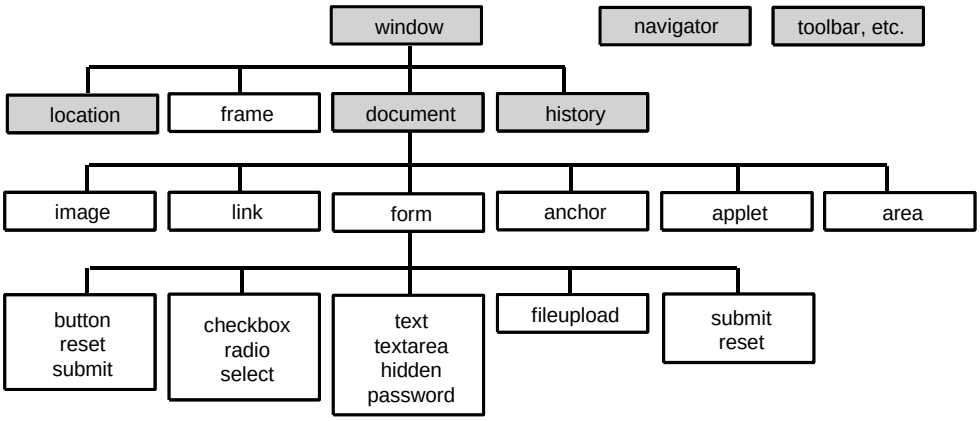
Even with standards to follow, it is left up to the browser vendors to implement the standards as they see fit, which in practice means spotty support and proprietary extensions. It is also up to Web page developers to code their pages to work with both current and past browsers.

The Notes client itself, as well as the Notes browser (not IE) that launches when you hit a URL (if the Notes browser is specified as your browser in the current *Location* document), differ in their support of any standard browser DOM.

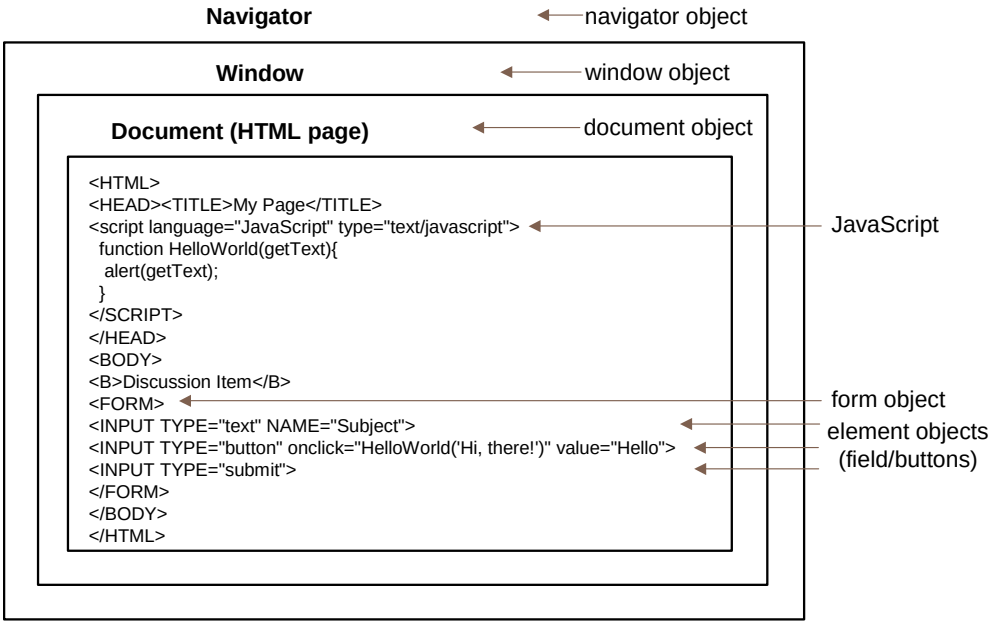
Note: The bulk of this course is based on DOM Level 0 and then turns to DOM Level 1 in a later Topic that discusses Dynamic HTML (DHTML).

Browser Object Model (DOM Level 0)

Virtually everything in a browser environment as well as the content on an HTML page is considered an object, with its associated events, properties, and methods. The shaded boxes in the browser DOM represent browser objects, while the unshaded boxes represent HTML objects that are created using HTML tags:



How does this relate to what a user sees on a Web page? This diagram shows the relationship between the HTML that makes up a Web page and the nested hierarchy of objects:



The Navigator object represents the browser itself. The Window object contains the Document object, which in turn contains elements, including one or more Form objects, each containing one or more Element objects.

JavaScript vis-a-vis browser objects

JavaScript has access to the methods and properties of browser objects. Although the various browsers differ slightly in their object models, there is a large common base of language, browser, and HTML objects that they all share.

The whole idea behind JavaScript is to create interactive Web pages that don't require a round trip to the server for server-side code execution. An equally important goal for dual-client applications is to provide equivalent features and experience for both audiences.

With its ability to work with these language, browser, and HTML objects, using JavaScript you can:

- automate navigation
- dynamically generate HTML content
- change the location and format of text and graphics using Dynamic HTML (the combination of HTML, Cascading Style Sheets, and JavaScript)
- show Field-level help in the browser status bar or a popup window as they hover over links
- prompt users for information using built-in alert, confirm, and prompt dialog boxes or through simulated @DialogBox or @Picklists with data fed from Notes documents or Views
- perform simple Field translation and validation interactively at the browser without refreshing the Form, which requires a round-trip and a page refresh (Domino runs the Field Input Translation and Validation formulas and returns a new page showing the results)
- change Form open behavior by calling a JavaScript function in the *onload* browser event handler; the Form submit behavior can be changed by calling a function in the *Submit* event handler
- create or destroy windows (such as to simulate custom dialog boxes)
- dynamically update window and Frame locations and content
- store user preferences in cookies on the local computer for later recall or passed as part of URL links for page pre-processing based on the preferences.

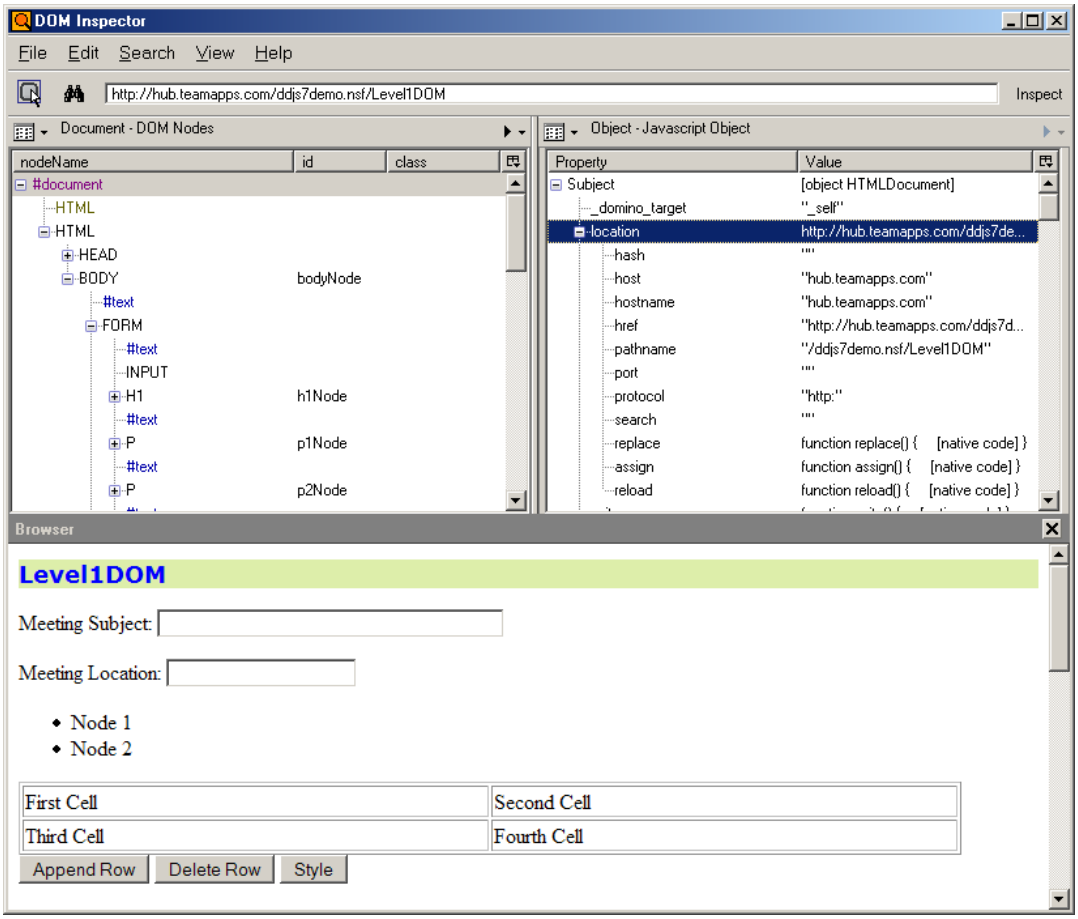
All of this is possible because JavaScript has full access to browser objects.

Note: Again, because Lotus Notes also supports JavaScript (though with fewer browser objects), the goal for the developer is to write applications that work for both browsers and Notes without recoding. In practice, you will find this possible for very basic things only, since Notes just doesn't completely support all the browser objects.

Mozilla's DOM and JavaScript object Inspector

Want to know everything there is to know about the DOM and JavaScript objects on a Web page? Open the page in the DOM Inspector that comes with Firefox 1.5 or Netscape Browser 8+!

To open the tool, choose **Tools –DOM Inspector**. In the left pane, you can navigate the DOM nodes and see the JavaScript object properties for that DOM node in the right pane:

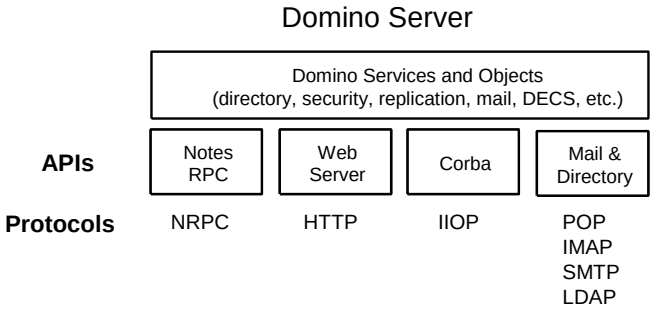


We think this tool will prove immensely helpful in understanding both the DOM and JavaScript object hierarchies. As the course progresses you will be working with both sets of objects.

Domino as a Web server

Domino Designer is what you use to create design elements that combine HTML and JavaScript, but it is the Domino Server that is responsible for serving the pages to browsers.

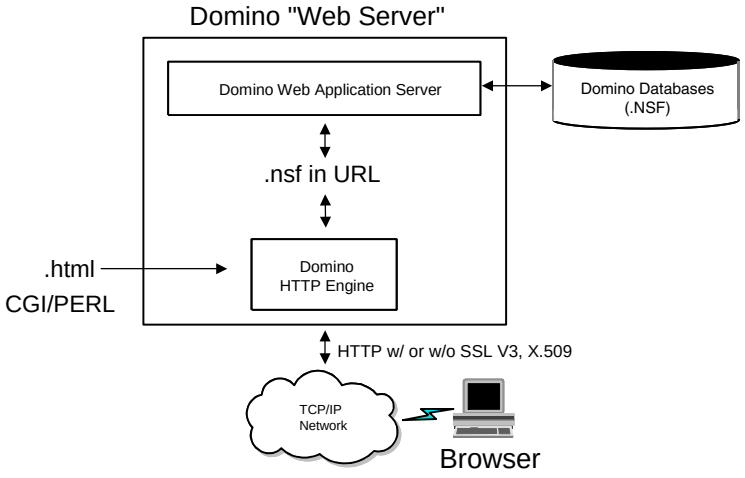
Domino is made up of a core executable and a number of add-in tasks to fulfill various functions, such as database access, replication, mail routing, etc. Together, these tasks contribute to the list of Domino Services and Objects, as shown in this diagram:



Domino provides client access via various protocols to various APIs. Browsers access the Domino services and objects through the Domino Web Server using the HTTP protocol.

The Domino "Web Server"

The Domino "Web Server" is made up of the Application Server and the HTTP engine. This diagram shows the relationship between the server components and the URL parsing performed by the HTTP server:



- **HTTP Server**. The native Domino HTTP server task is much like a typical HTTP server, in that it parses URL requests from browsers and interprets/returns pages (which contain HTML and client-side scripts) or refers calls to other interpreters (CGI/PERL) or servers (such as to streaming video servers). In addition to HTTP services, Domino also includes:
    - user authentication from the Domino Directory, X.509 certificates, or LDAP referrals
    - connection, server request, and cookie logging
    - URL mapping and redirection
    - authentication realms, single sign-on (between Domino Servers and between Domino and Websphere) and session-based authentication (you can create custom forms to enter name and password)
    - HTML file protection
    - Internet Cluster Manager for load balancing and failover
    - Domino Web Server API filters (DSAPI)
    - ability to run Java Servlets.
- **Application Server.** If the URL includes ".nsf", the commands appended to the URL (such as ?OpenForm) are passed to the application server for processing. The application server generates pages on-the-fly using the logic coded in the application and returns standard HTML back to the HTTP server to deliver to the browser.

Keep in mind that the HTTP server task and the application server both use the same Domino databases; as such, your applications, if designed correctly, can service both Notes clients as well as browser clients.