by
Mark
Gordon

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

Inside this article:

Related links:

Mark Judd: JavaScript Integration

Creating field help for your Domino applications

Domino Designer R5 Technical Overview

Domino and JavaScript: Dynamic Partners (Part 2)

Get the PDF:

JSinR5.PDF206Kb

Get Acrobat Reader

Ready to spice up your R5 applications with some JavaScript? Want to code your JavaScript directly within the new Domino Designer IDE? Write JavaScript that runs in both a Web browser and the Notes client? You can do all this with R5! Whether or not you've written any JavaScript in your Domino R4.6 applications, you'll be impressed by what you can do with JavaScript in R5, because R5 introduces broad support for JavaScript -- both for coding it in Domino Designer and running it in the Notes client.

JavaScript is now built into the integrated development environment (IDE) in Domino Designer R5. This means that you get syntax checking, a color-coded editor, and a "home" for your JavaScript code. This article introduces R5's support for JavaScript by first talking about *why* you might want to use JavaScript in your applications, and then providing a couple of step-by-step examples of how to use it. Specifically, the examples show you how to calculate field values on the fly, and how to handle field validation at the form level before a document is submitted. As you'll see, you can in many cases build a single feature using JavaScript and run it in both Web browsers and Notes clients -- what we call Code Once, Run Everywhere, or C.O.R.E! We'll also show you where to put your code, how to set up reusable functions for a form, and how to handle global variables on your form.

If you haven't already done so, we strongly urge you to read the **Iris Interview with Mark Judd**, Iris' self-proclaimed JavaScript bigot. Mark answers many of the common questions about JavaScript in R5, such as what version of JavaScript it supports, what the Document Object Model (DOM) is, how it works with dynamic HTML, and future plans for JavaScript in Notes.

Downloading the sample database

You can try out the techniques described in this article by downloading the following self-extracting database (52Kb):

JSinR5.exe

Note: This database requires Designer R5, which you can **download here**.

Why use JavaScript?

The most important reason for using JavaScript in R4.6 applications was to support dynamic client-side processing when designing for Web browsers. You could use JavaScript to enable client-side field calculations, keyword lookups, and pop-up windows. You could even do things like create a field help frame that displays help text when users tab into that field, as described in my previous *Iris Today* article, "**Creating field help for your Domino applications**."

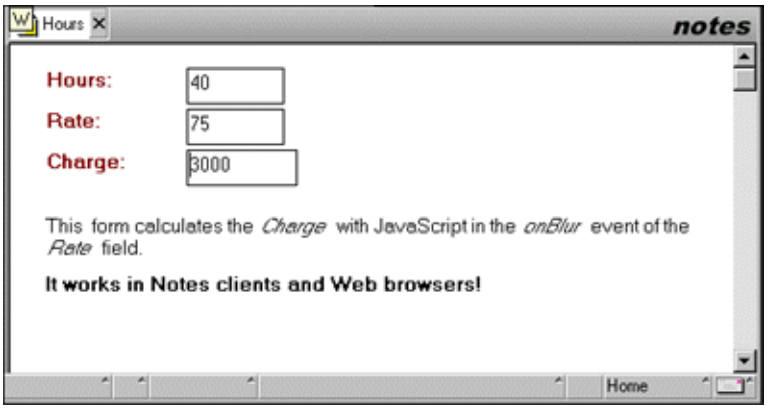
Now, with R5, there is another very compelling reason to use JavaScript: you can use it to help you design a single application that will run in both a Web browser *and* a Notes client. In R4.6, you had to write your JavaScript

in places the Notes client didn't use -- like right on the form (and hidden from Notes) or in a field's HTML attributes area. And you couldn't run your JavaScript in Notes client applications. But that's all changed with R5!

In R5, Domino Designer includes JavaScript support in the IDE, so you don't have to hide your code any longer. And JavaScript runs in the Notes client as well.

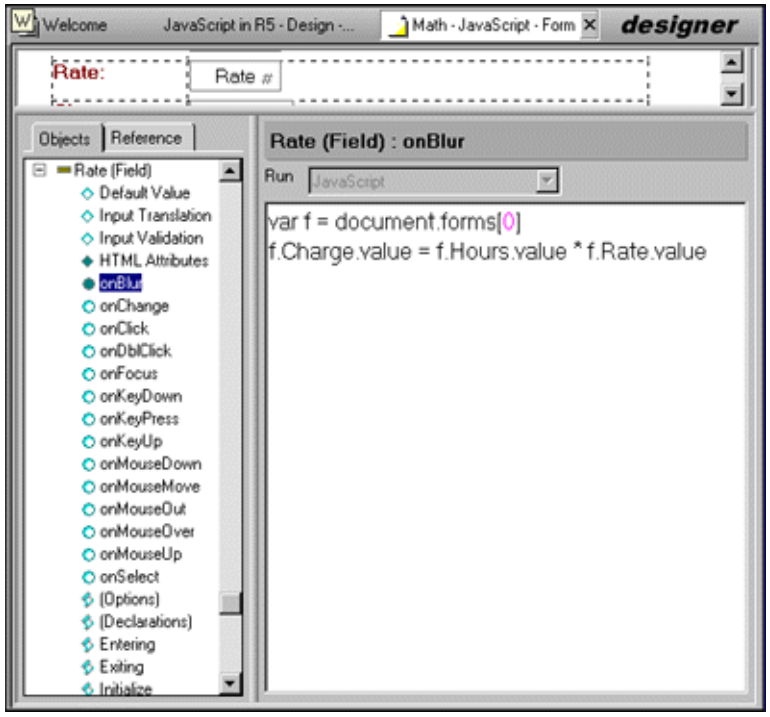
Using JavaScript for on-the-fly calculations

To begin our examination of JavaScript in R5, let's start with a simple example of calculating field values on the fly. In the sample database, the "Math - JavaScript" form calculates a charge by multiplying the hours and rate fields. As soon as you tab out of the Rate field, the Charge field calculates, as shown here running in the Notes client:



If this were strictly a Notes application, you could handle the calculation of the Charge field in a couple of ways: you could either put an input translation formula in the field -- Hours * Rate -- or you could put a LotusScript *field exiting* script behind the Rate field to multiply the two fields together. Then, to make the calculation dynamic for Web browsers in a Domino R4.6 application, you could write a JavaScript *onBlur* script for the Rate field.

Now, with R5, you can use JavaScript for a single solution that works for both Notes clients and Web browsers. Let's look at the same form in Domino Designer R5. When you click on the Rate field in the Objects tab of the new IDE, you see a whole list of events associated with that field. You can see formula language events (Default Value, Input Translation, and Input Validation), JavaScript events (such as onBlur, onChange, onClick, and so on), and LotusScript events (such as (Options), Entering, and Exiting). When you select an event, the combo box on the programmer's pane shows the language choices. In the following screen, we've selected the onBlur event, and JavaScript displays as the only language choice:



Notice the filled-in circle next to onBlur in the Objects tab. This tells us that we have written code for the event. Clicking on the onBlur event shows us the JavaScript in the programmer's pane:

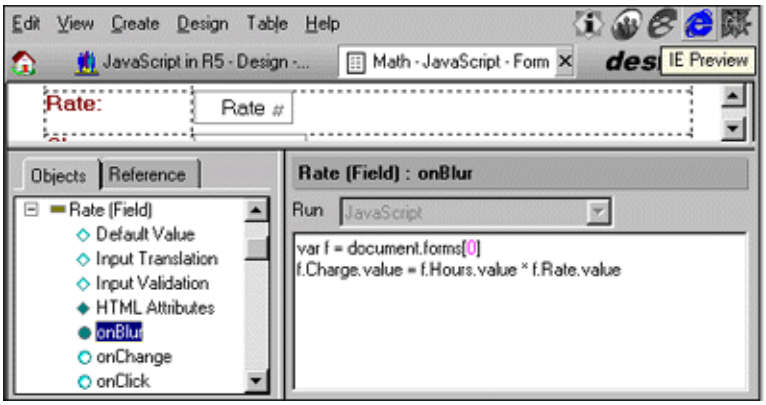
```
var f = document.forms[0]

f.Charge.value = f.Hours.value * f.Rate.value
```

This is a pretty simple piece of JavaScript. For those of you new to JavaScript, we're simply accessing the field values here via the JavaScript class hierarchy, much like you would access them via the Notes class hierarchy through LotusScript. The difference here is that we don't need to actually reference any Notes object classes, only JavaScript object model classes for a form -- classes that are the same in the browser. We've declared a variable *f* to reuse, but we could reference a field value directly via the class hierarchy, like this:

```
document.forms[0].Charge.value = document.forms[0].Hours.value *
document.forms[0].Rate.value
```

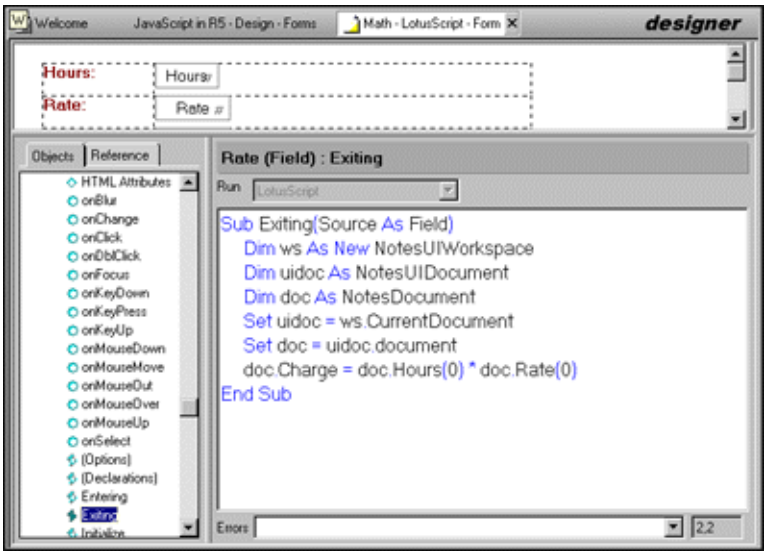
The document object refers to the JavaScript document object, *not* a Notes object such as NotesDocument or NotesUIDocument object. A document in JavaScript is basically a Web page. The document contains one or more forms, accessible via the *forrms* array, which is a property of the *document* object. So *document.forms[0]* is the first form on the "page." The *.Charge.value*, for example, refers to the value property of the Charge field on that form.

Remember that this JavaScript solution runs in the Notes R5 client as well as in Web browsers. So, you can preview this form in Notes, Domino (with the Domino browser), Internet Explorer, or Netscape. This screen shows the preview bar, with the cursor hovering over the IE preview icon:



Notice something else very nice about the JavaScript method of calculating the Charge: if you tab out of the Rate field and leave either the Hours or Rate fields blank, you won't get an error. If you used a formula language input translation formula for the calculation, like Hours * Rate, you would receive an error if either the Hours or Rate fields were empty.

Even if your application is geared only to Notes clients, the JavaScript solution is nicer than the formula language equivalent because the calculation occurs right when you tab out of the Rate field. The only way to do the same thing with the formula language is to set the form to recalculate all formulas every time you tab out of any field. Of course, you can do this in LotusScript. The following field exiting formula in the "Math - LotusScript" form would have the same effect as our JavaScript:



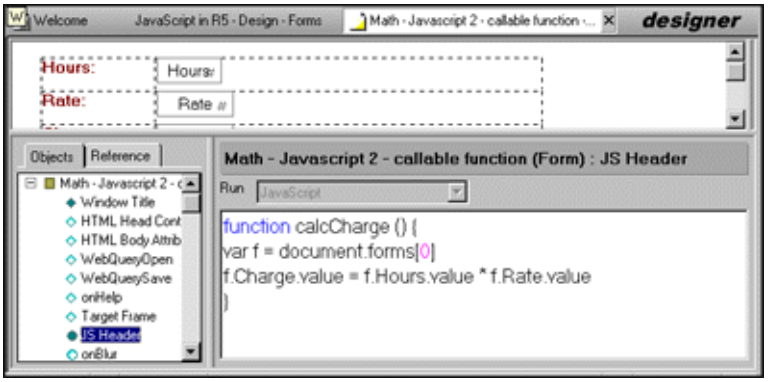
This works, but the JavaScript code is simpler than the LotusScript code, and it works both in a Web browser and the Notes client!

Defining reusable functions in the IDE

In our previous JavaScript example, we calculated the charge only when tabbing out of the Rate field. That works fine if you enter the hours, tab into the Rate field and enter your rate, and then press Tab. But what if you go back to the Hours field and enter a new value? It would be nice if the calculation would run again when you tab out of the Hours field. To do this, you could put the same calculation in both fields' onBlur events, of course, but especially with longer, more complex calculations, you would want to avoid having to maintain the same calculation in two places. (In addition, you'd probably want to call the calculation when users press the Submit

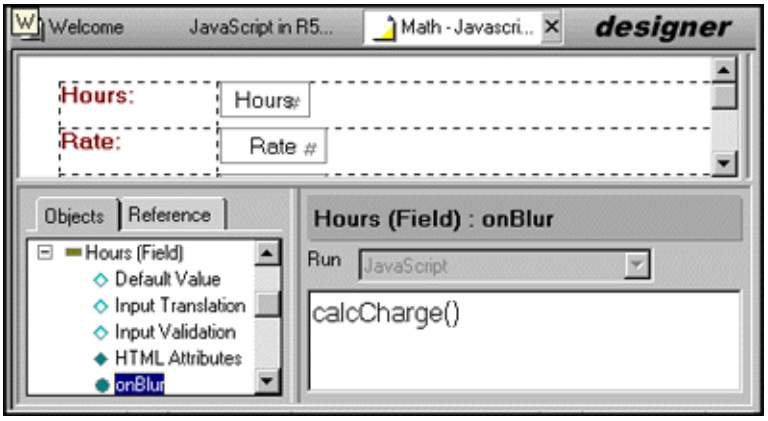
button, in case they hadn't actually tabbed out of the field they had just changed.)

Instead of copying code to multiple places, you can instead define a function, *calcHours*, which will multiply the two fields. You can then call that function from the onBlur event of both fields. In the sample database, the "Math - JavaScript2 - callable function" form includes this new calcHours function in the JSHeader event:



You can define any number of reusable functions in the JS Header event. (We'll look more at the JS Header event in the next section of this article.)

For the calculation, you can then simply place a call to the calcHours function in the onBlur event of both the Hours and Rate fields:



Now the Charge field recalculates when you tab out of either the Hours or Rate field. If you preview this form in a browser and view the source, you'll see that Domino generates the JavaScript in the JS Header within the <Head> and </Head> tags, like this:

```
<HTML>
<!-- Lotus-Domino (Release 5.0 - March 30, 1999 on Windows NT/Intel) -->
<HEAD>
<TITLE>Hours</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
function calcCharge () {
var f = document.forms[0]
f.Charge.value = f.Hours.value * f.Rate.value
}
// -->
</SCRIPT>
</HEAD>
<BODY TEXT="000000" BGCOLOR="FFFFFF">
```

```
<FORM METHOD=post
ACTION="/JavaScriptinR5.nsf/687cf3f585b3ec4f05256711004046b9?
CreateDocument" NAME="_DominoForm">
. . .
. . .
. . .
```

Next, let's look at the JS Header and why you'd want to use it.

Global variables, inline script and the JS Header

Those of you who have written JavaScript before are probably familiar with "inline" script -- JavaScript that you place directly on an HTML page, between <Script> and </Script> tags. In Domino R4.6, which had no JavaScript IDE, the Notes form was the logical place to put much of your JavaScript code (as pass-thru HTML). You could put reusable functions there, as well as inline script -- declarations for global variables and other JavaScript that executed as the page loaded. You'd manually place all of this between <Script> and </Script> tags on the form.

In R5, the IDE's JS Header is the place for your reusable functions, but it's not inline script, so be careful what you put there! There is currently no place for inline script in the R5 IDE. Domino places JS Header code into the <HEAD> section rather than as inline script, so simply declaring a global variable (such as, *var f*) still works. However, you can't assign the variable at the same time that you declare it, like many people did in R4.6 (such as, *var f = document.forms[0]*). The reason is that when the JavaScript executes in the <HEAD> section, the document object has not yet loaded. So, the object is not yet available to JavaScript.

In our calcHours function, we declared a variable *f* to reference the current form, so that we could refer to *f* instead of *document.forms[0]* each time we accessed a field on the form.

We *declared*, but did not *assign* the variable in the JS Header, like this:

```
var f;
```

We then assigned it in the form's onLoad event:

```
f = document.forms[0]
```

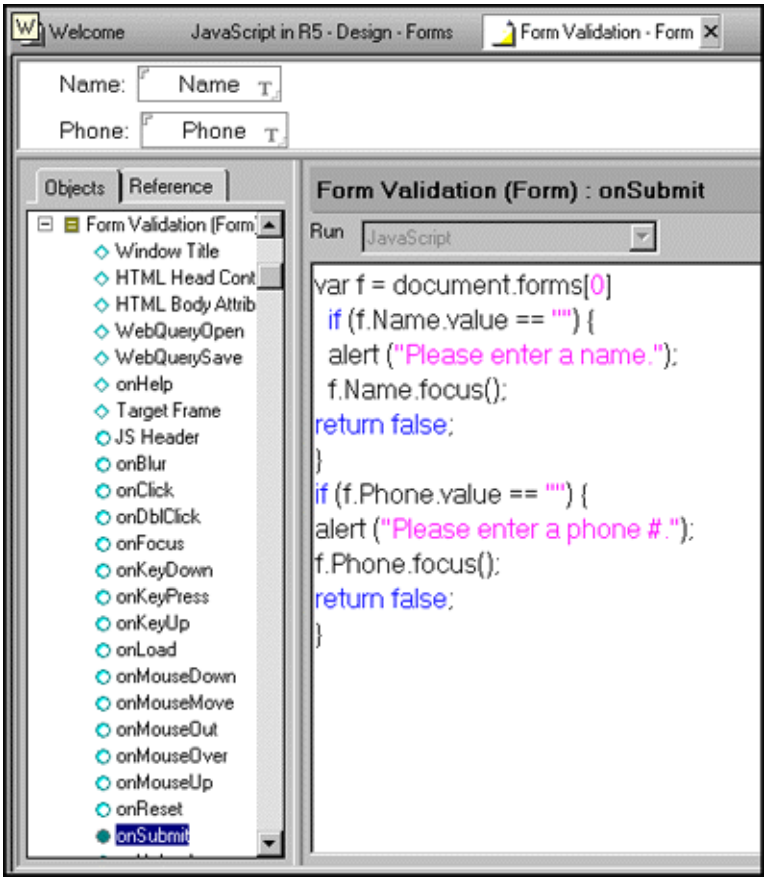
This accomplished the same thing as the inline script:

```
var f = document.forms[0].
```

Using JavaScript for field validation

Validating user input is something that JavaScript does very well for Web users. Rather than having to wait until another page loads to tell them what fields they missed, users can see an alert box -- much like they would with Notes input validation formulas.

Now you can do your field validation once, in JavaScript, for both Notes and Web clients! Take a look at the Form Validation form in the sample database. The onSubmit event checks to see if the Name field is blank, and if it is, returns *false.*



Here is the JavaScript:

```
var f = document.forms[0]
  if (f.Name.value == "") {
  alert ("Please enter a name.");
  f.Name.focus();
return false;
}
if (f.Phone.value == "") {
alert ("Please enter a phone #.");
f.Phone.focus();
return false;
}
```

Like a *continue = false* in LotusScript, returning *false* from the onSubmit event prevents the document from being saved. This works whether in the Notes client or a browser.

What remains different between the Notes client and a Web browser is how you "submit" a document. A Web *submit* is roughly equivalent to a Notes save-and-close. The only difference is that in Notes, after the window closes, the user is returned to the window that was open before the document was composed (or edited, in the case of an existing document).

To provide the correct "submit" functionality for Notes users, you can simply add a "Save and Close" action button, and hide it from Web users. In our Form Validation form, the Save and Close button contains the standard save-and-close formula:

```
@If (
  @Command ([FileSave]);
```

```
    @Command ([FileCloseWindow]);
 NULL
)
```

Browser users see only the Submit button. Either way, the onSubmit code is executed. The user experience is the same from the Notes client or Web browser, and is basically the same experience that Notes users would have if you used standard Notes input validation formulas for field validation.

Checkboxes and radio buttons make life harder

Simply checking whether a text field has been left blank is pretty easy. What's harder, of course, is validating a field that has been filled in, if you're looking for phone numbers or social security numbers, for example. Fortunately, there are lots of good sample JavaScript libraries available on the Internet. (For example, Netscape has **sample JavaScript**, as do other sites.) What makes life harder are radio buttons, checkboxes, and select boxes (listboxes and combo boxes). Unlike with the formula language or LotusScript, the JavaScript object model does not let you simply check to see if a radio button or checkbox field has a value. For example, the following code (from the "Form Validation - Include Radio Buttons" form in the sample database) shows how you must loop through each choice in a set of radio buttons to see if one is selected. This JavaScript code appears in the onSubmit event:

```
var f = document.forms[0]
  if (f.Name.value == "") {
  alert ("Please enter a name.");
  f.Name.focus();
return false;
}
if (f.Phone.value == "") {
alert ("Please enter a phone #.");
f.Phone.focus();
return false;
}
// Now check to see that one of the radio buttons is chosen
rb = false;   // assume none chosen until specified otherwise
for (i = 0; i < f.ChooseOne.length; i ++) {
  if (f.ChooseOne[i].checked) {
    rb = true;   // as long as one is chosen this gets set true
  }
}
if (rb == false) {
    alert ("Please choose one");
    f.ChooseOne[0].focus();
    return false;
}
```

Select boxes and checkboxes are slightly different from radio buttons, and to make matters worse, the object models vary from browser to browser and version to version. For example, in Netscape Navigator, if more than one checkbox is defined, the object contains an array of selection pointers to an array of strings containing the keyword values, and the *checkbox.checked* property contains the keyword. But if only one checkbox exists for a field, the keyword value is a string. And, in Internet Explorer, the values are an array.

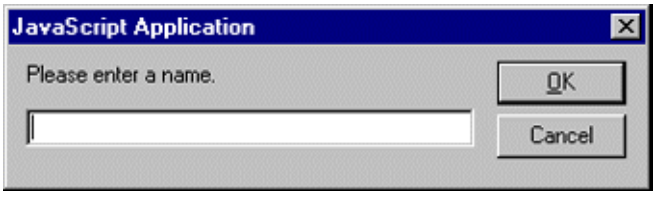
For an excellent discussion of field validation amongst various browsers, see the article "**Domino and JavaScript: Dynamic Partners (Part 2)**." The article includes working validation routines that handle everything: text fields, checkboxes, and radio buttons for the R3.x and R4.x versions of

Netscape and Internet Explorer. What you'll need to add is a check for the Notes R5 client.

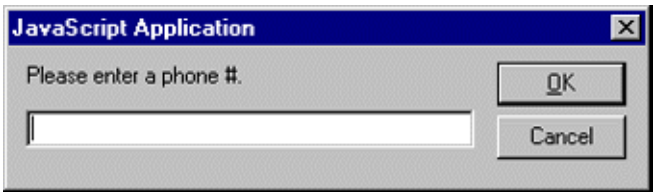
Even better field validation -- users like this approach better

The problem with the previous examples is that users who unintentionally leave several required fields blank on a form can end up getting an alert box four or five times. Each time they fill in a field and click the Submit button again, they receive another alert! Whether the prompts are coming from JavaScript alert boxes or Notes client input validation formulas, this type of field validation can be very frustrating for the user.

A much more user-friendly way of validating fields is to prompt users to enter anything they missed, forcing the input before the submit is made, but not cancelling the submit. For example, let's say that your form includes a Name field and a Phone Number field. If users leave both fields blank, they receive two prompts, one after the other. This one:



followed by this one:



When users first click the Submit button on the form, they receive the prompt for their name. After entering a name and clicking OK, users don't have to click the Submit button again. Instead, the second prompt displays. After entering a phone number and clicking OK, the form is automatically submitted. (It also runs true validation, just in case the user clicked Cancel on one of the prompts and still left a field blank.) This user experience may not seem very different, but it is actually a lot less frustrating than for the user needing to continually click Submit.

Many Notes developers have been giving users this more user-friendly approach to field validation for years. Now, you can do it with JavaScript, and it works for both browsers and Notes clients.

Here's how the *onSubmit* handler would look for the Name field:

```
var f = document.forms[0];
// Check Name field
  if (f.Name.value == "" || f.Name.value == "null") {
  msg = "Please enter a name.";
  // Prompt user to enter the name
  f.Name.value = prompt (msg, "");
  // If they hit cancel (returns "null") or still leave it blank, alert and
return false
  if (f.Name.value == "" || f.Name.value == "null") {
    f.Name.value = ""
    alert (msg);
  f.Name.focus();
return false;
  }
```

```
}
```

Conclusion

Now you have an overview of how JavaScript works in R5, as well as a couple of practical examples for how to use it in your own applications. You can now use JavaScript to make your design work the same way for both Notes and Web clients. Future releases promise to bring more JavaScript features, so you can do even more things to Code Once, and Run Everywhere!