Debugging with Venkman

Page 2 — Stepping up

There are three main steps to debugging JavaScript with Venkman. First, we make the debugger stop where we think the problem lies, then we step through the code and observe it to see exactly what's going wrong. Finally, we fix the problem.

The best way to jump into the debugger is to set a breakpoint. A breakpoint simply says, "When we reach this line, start debugging." Let's set one now.

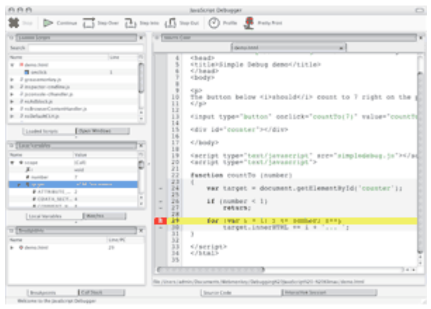
Open our demo page and fire up Venkman. Look for demo.html in the Loaded Scripts pane. Look inside the file by clicking on the disclosure triangle and find the countTo function. Double-click that to pull it up in the Source Code pane.

See how in the left margin of the Source Code, there are dashes? Click the one beside the line that reads:

```
for (var i = 1; i <= number; i++)
```

The dash turns into a bright red B, indicating you've set a breakpoint. You can click it two more times to clear the breakpoint. (It toggles to an F along the way — this indicates a special kind of breakpoint that you don't need to worry about just yet.) But leave it be for now and switch back to the regular browser window. Click the countTo button and it'll throw you back to the debugger, only now you've frozen execution.

Notice that the Local Variables pane now shows our variables: i, number, and target. Because "target" is an object, you can dig into its properties like you did the source code files.


```
<head>
<title>Simple Debug demo</title>
</head>
<body>

<p>
The button below <i>should</i> count to 7 right on the p
</p>

<input type="button" onclick="countTo(7)" value="countTo

<div id="counter"></div>

</body>

<script type="text/javascript" src="simpledebug.js"></sc
<script type="text/javascript">

function countTo (number)
{
    var target = document.getElementById('counter');

    if (number < 1)
        return;

    for (var i = 1; i <= number; i++)
        target.innerHTML += i + '... ';
}

</script>
</html>
```

Venkman displaying object properties (Take a closer look)

You can use the debugger's toolbar to control the flow of execution. The Continue button stops debugging entirely, and runs the rest of the script. The three Step buttons work in similar ways — they basically control whether you dive into subfunctions or not. Step Over skips past a subfunction call, while Step Into delves deeper. If you get too deep into things, use Step Out to jump up the next highest function call. If you click Step Out at a function that was invoked directly by the user, it acts just like the Continue button.

So go ahead and click Step Into a bunch of times. Watch how the i value changes as you loop through the for loop. Pretty nifty, eh? You don't have to just sit there. Right-click the value of number and choose Change Value. Type in `14` and hit OK, then click Continue in the debugger toolbar. Indeed, it counts to 14 instead of 7. You can use this trick to simulate weird conditions in your script, or just to experiment to see how your script reacts.

Digging through an object to find a particular property can be cumbersome. Watches provide shortcuts to exactly the sliver of information you want to "watch" for. Right-click the empty space in the Watches pane and choose Add Watch Expression. Type in `target.innerHTML` and hit OK. When you step through the countTo function again, you can watch innerHTML get written to.

Watches can evaluate any kind of expression at all. Add another watch, but this time type `(i == number)`. The debugger tells us it's false. This is a pretty simple condition, but you could use similar ones to track all kinds of things while you step through your code. You can even save a set of watches for future reference. Just right-click the Values pane and choose Save Watch Settings. The item below it, Restore Break/Watch Settings, gets them back. Incidentally, there is a matching Save menu item for the Breakpoints pane.

Now that I've given you some toys to play with, go ahead and try them out with your own code. For more information on Venkman, try the official FAQ or even this guide, which goes into more detail than I have space for here.

Happy debugging!

Did you love this article? Did you hate it? Think you can do better? Send us your Feedback. Feedback submitted here will be considered for publication on Webmonkey or Wired News, so if you don't want us to print your comments, please say so in your email.

[an error occurred while processing this directive] [an error occurred while processing this directive]