SAND99-1006J

SAND99-1006J

Server-Side JavaScript Debugging: Viewing the Contents of an Object

Randall Simons and Jeff Hampton
Sandia National Laboratories

JavaScript allows the definition and use of large, complex objects. Unlike some other object-oriented languages, it also allows run-time modifications not only of the values of object components, but also of the very structure of the object itself. This feature is powerful and sometimes very convenient, but it can be difficult to keep track of the object's structure and values throughout program execution. What's needed is a simple way to view the current state of an object at any point during execution.

There is a *debug* function that is included in the Netscape server-side JavaScript environment. The function outputs the value(s) of the expression given as the argument to the function in the JavaScript Application Manager's debug window [SSJS]. For example, the following lines in Figure 1 of a server-side JavaScript program:

```
function myObj() {
  this.myVal = 1;
  this.myArr = new Array(2, 3);
  this.myStr = new String("test");
  this.myFunc = myFunc;
}
function myFunc(arg1, arg2) {
  return(arg1 - arg2);
}
var x = new myObj();
debug("x.myVal = ", x.myVal);
debug("x.myArr[0] = ", x.myArr[0]);
debug("x.myStr = ", x.myStr);
debug("x.myFunc = ", x.myFunc);
debug("sum = ", x.myVal + x.myArr[1]);
debug("x = ", x);
```


Figure 1. Example code.

produce the output in Figure 2:

```
Debug message: x.myVal = 1
Debug message: x.myArr[0] = 2
Debug message: x.myStr = test
Debug message: x.myFunc = function myFunc(arg1, arg2) { return(arg1 - arg2); }
Debug message: sum = 4
Debug message: x = [object Object]
```

Figure 2. Example code output.

This function is useful for checking the values of individual variables or expressions. But it doesn't tell you much about more complex data structures, such as the object variable 'x' in the example. It would be more useful to see the values of all elements in an array or properties in an object. (For purposes of this discussion, an array can be considered a type of object. When the word "object" appears below, interpret that to include arrays also.) To aid us in viewing object structure in our programs, we have written a recursive function called *dumpObj* which steps through all the properties in an object, outputting a type, name, and value for each. Using the same example object above, a function call such as the one in Figure 3:

DISCLAIMER

```
dumpObj("x", x);
```

Figure 3. Example *dumpObj* function call.

would produce the output in Figure 4:

```
Debug message: number: x.myVal = 1
Debug message: number: x.myArr.0 = 2
Debug message: number: x.myArr.1 = 3
Debug message: String: x.myStr = test
Debug message: function: x.myFunc = myFunc(arg1, arg2)
```

Figure 4. Example *dumpObj* function output.

The code for *dumpObj Function Ver. 1.0* is presented in Figure 5:

```
// dumpObj Function Ver. 1.0
// Purpose: Write out the structure and values of an object.
//
// Arguments:
// name = a string to label the output with.
// obj = an object to output the values of (simple variables are OK).
// depth = number of levels to descend in object (default = 10).
//
// Example: dumpObj("this", this, 4);
//
// Author: Randall W. Simons
// Copyright 1999 Sandia Corporation.  Under the terms of Contract
// DE-AC04-94AL85000, there is a non-exclusive license for use of
// this work by or on behalf of the U.S. Government.
//
function dumpObj(name, obj, depth) {
  if(obj == null) return;                    // Ignore null values.
  var dep = 10;
  if(depth != null) dep = depth;
  var cons = obj.constructor;
  var type = typeof obj;
  if(cons == String) {
    // Output string on one line, not as array of characters.
    debug("String: " +name+ " = " +obj);
  }
  else if(type == "function") {
    // Format function value to output only function name and arguments.
    var val = new String(eval(obj));
    val = val.substring(9, val.indexOf(")")+1);
    debug(type+ ": " +name+ " = " +val);
  }
  else if(type == "object" ) {
    if(dep < 1) {
      // Cut off recursion when depth reaches limit.
      debug(type+ ": " +name+ " = " +obj);
    }
    else {
      // Recursively call dumpObj for every element in obj.
      for(sub in obj) {
        dumpObj(name+ "." +sub, obj[sub], dep-1);
      }
```

```
    }
  }
  else {
    // If not one of the special cases above, output a simple variable.
    debug(type+ ": " +name+ " = " +obj);
  }
}
```

Figure 5. *dumpObj Function Ver. 1.0* code.

The *dumpObj* function has three arguments:

Name: Is an argument that represents a character string to put at the beginning of the line after "Debug message: *Type*:". This is typically the name of the object whose values are to be dumped. You may also want to include the name of the function that *dumpObj* is being called from, to make it easy to identify which debug messages come from where.

Obj: Is the variable name of the object whose values are to be dumped.

Depth: Is an optional parameter that limits how many levels deep to show an object's properties. The default is 10 levels deep. A depth of 1 results in essentially the same output as if you just used the *debug* function, and only tells you what you have is an object or array. A depth of 2 outputs all the properties in the object, but doesn't break down those properties further.

Caution: If the object passed to *dumpObj* is too large, the server-side JavaScript interpreter may produce an error. If this happens, "prune" the object using the depth parameter, or just specify one piece of the object to output.

Objects with a constructor of *String* have to be treated as a special case because JavaScript stores them as character arrays. Thus, the *dumpObj* function would continue to break down the string into individual characters, and print each in a separate debug message line similar to the way normal arrays would be output by the function. The *dumpObj* function looks for that type of variable and just prints the string out in its entirety instead to make the output more readable. The *dumpObj* output for functions is also modified so the function body is not shown, making the output more readable.

The *dumpObj* function is a versatile tool to have in your programming toolbox. By adding a single *dumpObj* function call to your code, you can see all the details of an object, no matter how large and complex. The power of recursion keeps the *dumpObj* code surprisingly simple.

References

[SSJS] Netscape Communications Corporation, "Server-Side JavaScript Guide v1.2", 1998, ch. 3, section. "Debugging an Application", http://developer.netscape.com/docs/manuals/js/server/jsguide2/appdev.htm#1046343

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.