Server Side Scripting JavaScript

History

JavaScript is not only applicable to the front-end side of scripting but also in a server-side perspective. JavaScript as a server-side scripting language does not involve any downloading of scripts from the browser because it is accessed directly through the server. The first implementation of Server-Side JavaScript (SSJS) was in Netscape’s LiveWire, which in 1996 included in the Enterprise Server 2.0. After the birth of the server-side JavaScript, a lot of companies tried their own alternative or version of server-side technologies in accordance to the existing server-side scripting technologies. Such examples are Microsoft’s ASP, VBScript Language which also supports JavaScript and Perl Script. Microsoft has also their own version of Javascript which is JScript. To use either JavaScript or JScript, you have modify the opening script tag and set the language to the type of Javascript you would want to target.

Sample code:

```
<%@LANGUAGE="JavaScript"%>
<%
Response.Write("<HTML>\r")
Response.Write("<FONT COLOR=\"red\">\"Hello World\"</FONT><BR>\r")
Response.Write("</HTML>\r")
%>
```

Since web server run the code for clients, the output displayed on the client side is the projected output of the script and not the source code. Therefore, only the Response.write functions will display the output on the source.

Sample code:

```
<HTML>
<FONT COLOR="red">"Hello World"</FONT><BR>
</HTML>
```

Introduction

JavaScript has traditionally run in browsers only. Other server-side JavaScript environments include Jaxer and Narwhal. However, Node.js, other than the other server-side JavaScript technologies, if offers event-based rather than thread-based. Thread based can induce a system thread for all requests made. Most web servers like Apache that are using PHP and other CGI scripts to serve it.

Node.js, on the other hand, instead of implementing threads, it uses event loop and to scale connections ranging from hundred to millions. The role of most servers are to wait until it has its time to receive requests for I/O operations. Every I/O operation in Node.js is asynchronous, even though there are multiple requests, the server can continue responding to requests.JavaScript has functions and closures which is really suited to event-based programming and also the concept of callbacks and etc.

- **Concepts on JavaScript as Server-Side Scripting**

Introduction

You can execute Server Side JavaScript (SSJS) in Content Containers and Design Parse Files by including the runat="server" attribute when using a <script> tag. For example:

```
<script runat="server">
  print('<h1>%asset_name%</h1>');
</script>
```

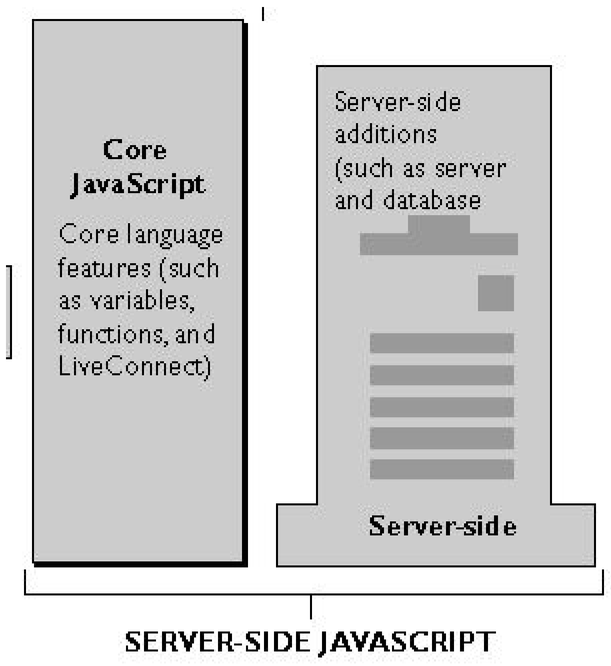
You can also execute JavaScript from a JS File by sourcing it using the src attribute, for example:

```
<script runat="server" src="./?a=XXXX"></script>
```

This functionality is similar to that of the JavaScript Processing field on the REST Resource JS asset.

Note: When used in Content Containers, SSJS is not processed at the time of the processing of the Content Container content. Instead, it is processed at the global level along with any Global Keyword Replacements.

Server-Side JavaScript Object Model

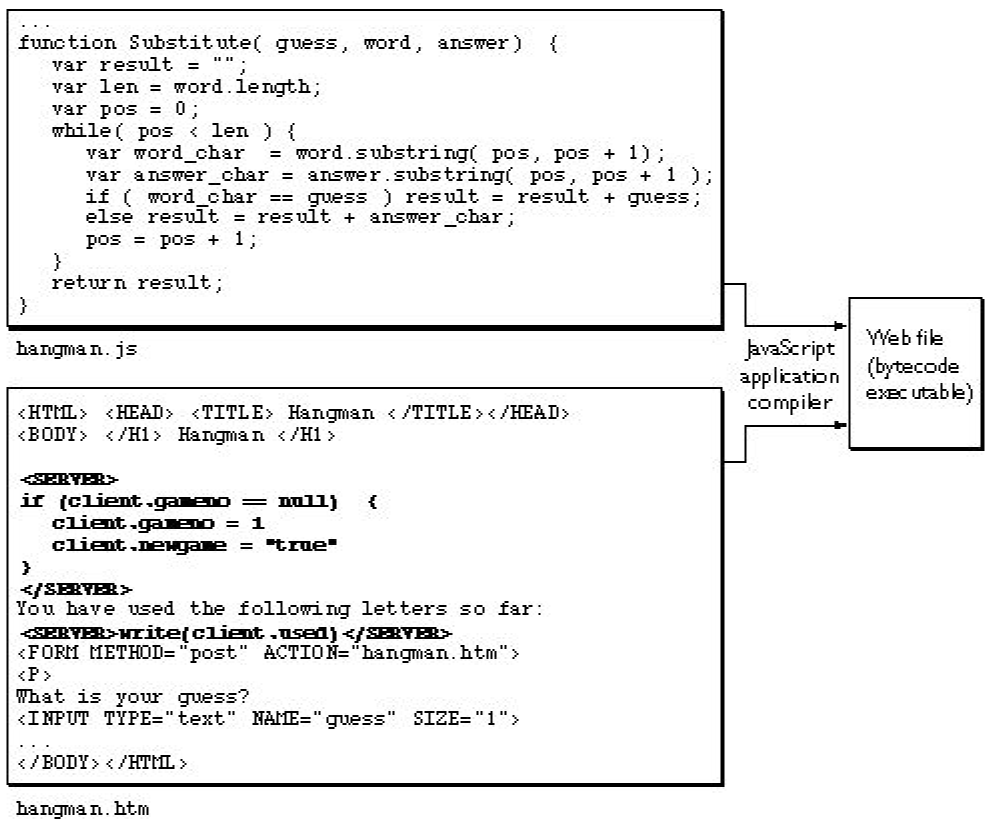

Core JavaScript
Core language features (such as variables, functions, and LiveConnect)
Server-side additions (such as server and database
Server-side
SERVER-SIDE JAVASCRIPT

On the server, you also embed JavaScript in HTML pages. The server-side statements can connect to relational databases from different vendors, share information across users of an application, access the file system on the server, or communicate with other applications through LiveConnect and Java. HTML pages with server-side JavaScript can also include client-side JavaScript.

In contrast to pure client-side JavaScript pages, HTML pages that use server-side JavaScript are compiled into bytecode executable files. These application executables are run by a web server that contains the JavaScript runtime engine. For this reason, creating JavaScript applications is a two-stage process.

In the first stage, shown in Figure 1.3, you create HTML pages (which can contain both client-side and server-side JavaScript statements) and JavaScript files. You then compile all of those files into a single executable.

Figure 1.3 Server-side JavaScript during development



```
...
function Substitute( guess, word, answer)  {
   var result = "";
   var len = word.length;
   var pos = 0;
   while( pos < len ) {
      var word_char  = word.substring( pos, pos + 1);
      var answer_char = answer.substring( pos, pos + 1 );
      if ( word_char == guess ) result = result + guess;
      else result = result + answer_char;
      pos = pos + 1;
   }
   return result;
}
```

hangman.js

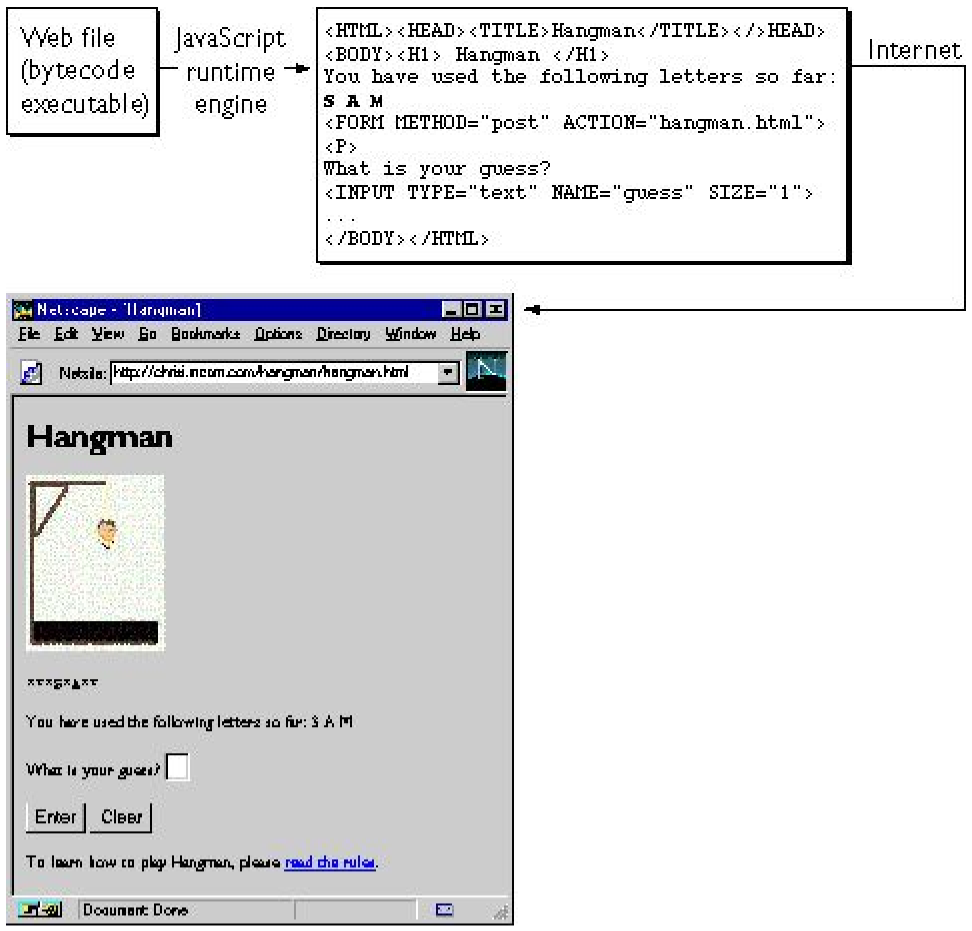
```
<HTML> <HEAD> <TITLE> Hangman </TITLE></HEAD>
<BODY> </H1> Hangman </H1>

<SERVER>
if (client.gameno == null)  {
   client.gameno = 1
   client.newgame = "true"
}
</SERVER>
You have used the following letters so far:
<SERVER>write(client.used)</SERVER>
<FORM METHOD="post" ACTION="hangman.htm">
<P>
What is your guess?
<INPUT TYPE="text" NAME="guess" SIZE="1">
...
</BODY></HTML>
```

hangman.htm

JavaScript application compiler
Web file (bytecode executable)

In the second stage, shown in Figure 1.4, a page in the application is requested by a client browser. The runtime engine uses the application executable to look up the source page and dynamically generate the HTML page to return. It runs any server-side JavaScript statements found on the page. The result of those statements might add new HTML or client-side JavaScript statements to the HTML page. The run-time engine then sends the resulting page over the network to the Navigator client, which runs any client-side JavaScript and displays the results.

Figure 1.4   Server-side JavaScript during runtime


Web file (bytecode executable)
JavaScript runtime engine
Internet
Netscape - [Hangman]
File Edit View Go Bookmarks Options Directory Window Help
Netsite: http://chris.mcom.com/hangman/hangman.html
Enter
Clear
Document: Done

```
<HTML><HEAD><TITLE>Hangman</TITLE></>HEAD>
<BODY><H1> Hangman </H1>
You have used the following letters so far:
S A M
<FORM METHOD="post" ACTION="hangman.html">
<P>
What is your guess?
<INPUT TYPE="text" NAME="guess" SIZE="1">
...
</BODY></HTML>
```

Hangman

You have used the following letters so far: S A M

What is your guess?

To learn how to play Hangman, please read the rules.

In contrast to standard Common Gateway Interface (CGI) programs, all JavaScript source is integrated directly into HTML pages, facilitating rapid development and easy maintenance. Server-side JavaScript's Session Management Service contains objects you can use to maintain data that persists across client requests, multiple clients, and multiple applications. Server-side JavaScript's LiveWire Database Service provides objects for database access that serve as an interface to Structured Query Language (SQL) database servers.

Global Variables

You can set global JS variables which can be accessed by other Content Containers in the page processing

```
<script runat="server">
  var userName = '%globals_user_attribute_username%';
</script>
```

I can then access the same global variable at any other point in the Matrix page generation, either in another Content Container or in the Design Parse File code:

```
<script runat="server">
  print(userName);
</script>
```

Attributes

You can pass additional attributes into the <script> tag for the server side functionality including:

runat="server" - Enables the JavaScript code within the tag to run server side.
evalkeywords="post" - By default, keywords are evaluated before the JS is processed. Adding this attribute will make the keywords evaluate after the JS has been processed.
This only applies to Global Keywords. In-context asset specific keywords such as %asset_name% will always need to be evaluated in the context of where they are printed.
log="log_prefix" - Adding this attribute will log any output from the JavaScript as an entry into the Matrix System log. The value of the log attribute will prefix the messages that are printed via the JS print() function.tributes.

Disabling SSJS

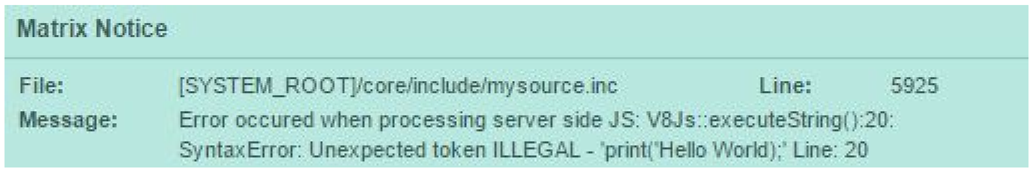
You are able to disable all JavaScript from running on the server using the runat="server" method, by adding the following query string parameter to the URL:
?SQ_DISABLE_SERVER_JS
The JavaScript in the Content Container will print within the HTML source and execute client side as normal JavaScript. This allows debugging of the JavaScript within the source code of the page.
You need to have Write Access to the Content Container for this query string to take effect.

Viewing SSJS

If you have Write Access to an asset, you can append ?SQ_VIEW_SERVER_JS to the URL when previewing it on the front-end in order to see a full output of the JS code that was sent to the JS engine in Matrix.

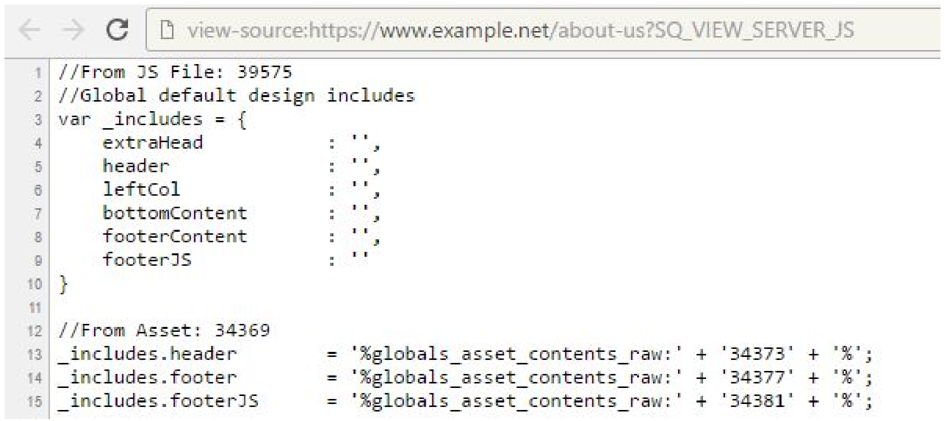
This is useful if you are getting JS errors on the front-end that relate to a specific line number in your code, as you will be able to track down exactly which line the error is referring to.

For example, let's say you got an error similar to the one in the example below. The error points to line 20 in the JS code.



```
Matrix Notice
File:      [SYSTEM_ROOT]/core/include/mysource.inc          Line:    5925
Message:   Error occured when processing server side JS: V8Js::executeString():20:
           SyntaxError: Unexpected token ILLEGAL - 'print('Hello World);' Line: 20
```

Viewing the front-end page with ?SQ_VIEW_SERVER_JS appended to the URL, you can see which line the error is referring to. You can also use JS comments to help indicate which specific asset each JavaScript block is coming from.



view-source:https://www.example.net/about-us?SQ_VIEW_SERVER_JS

```
 1  //From JS File: 39575
 2  //Global default design includes
 3  var _includes = {
 4      extraHead          : '',
 5      header             : '',
 6      leftCol            : '',
 7      bottomContent      : '',
 8      footerContent      : '',
 9      footerJS           : ''
10  }
11
12  //From Asset: 34369
13  _includes.header        = '%globals_asset_contents_raw:' + '34373' + '%';
14  _includes.footer        = '%globals_asset_contents_raw:' + '34377' + '%';
15  _includes.footerJS      = '%globals_asset_contents_raw:' + '34381' + '%';
```

Basic Example

This example will conditionally print a <metadata> SEO based section tag based on the value of the current frontend URL the user is viewing.

```
<script runat="server">
  //Create a variable with the current frontend asset's URL
  var frontendURL = '%frontend_asset_url%';
  //Create a variable for the section string
  var section = '';
  //Set the section string based on the section where the current URL is
located under
  if(frontendURL.indexOf('/manuals') > 0){
    section = 'Manuals';
  }else if(frontendURL.indexOf('/tutorials') > 0){
    section = 'Tutorials';
  }else if(frontendURL.indexOf('/news') > 0){
    section = 'News';
  }
  //Print a metadata tag for the current section if it is not empty
  if(section != ''){
    print('<meta property="article:section" content="' + section + '"/>');
  }
</script>
```

This SSJS code could then be placed somewhere inside the <head> tag of the page design.

- ### Server-Side JavaScript (SSJS) via Embedded JavaScript Engines

The extension of JavaScript to the server is made possible via embedded JavaScript engines. The two most well established engines are SpiderMonkey and Rhino, both currently maintained by the Mozilla Foundation. SpiderMonkey is the code name for the first ever JavaScript engine, an open source C implementation which can be found embedded in leading software products such as Mozilla Firefox, Adobe Acrobat, and Aptana Jaxer. Rhino is a Java implementation of JavaScript which is commonly embedded in Java applications to expose scripting capability. The Helma web application framework is an example of Rhino in use. The Dojo Toolkit also has capabilities that run in Rhino. Let's take a closer look at Rhino and its importance to the JavaScript developer.

Rhino offers a unique opportunity for the JavaScript developer to tap into the power of Java classes using JavaScript. Armed with some basic Java knowledge, you can extend JavaScript to include some of the most desired capability such as database access, remote web requests, and XML processing. We'll start by taking a look at querying a SQL database.

This first example we'll demonstrate is querying a mySQL database for some employee contact information. It is assumed that you have already downloaded, extracted, and consumed the necessary documentation to get up and running with some basic Rhino scripts. If you don't already have it, you'll need to also download the JDBC driver for mySQL, extract the class files, and include the path in your CLASSPATH environment variable. The code in Listing 1 is a sample JavaScript script which incorporates Java classes to handle the database query.

Sample Code:

```
// import the java sql packages
importPackage( java.sql );
// load the mySQL driver
java.lang.Class.forName( "com.mysql.jdbc.Driver" );
// create connection to the database
var conn = DriverManager.getConnection( "jdbc:mysql://localhost:3306/rhino",
"uRhino", "pRhino" );
// create a statement handle
var stmt = conn.createStatement();
// get a resultset
var rs = stmt.executeQuery( "select * from employee" );
// get the metadata from the resultset
var meta = rs.getMetaData();
// loop over the records, dump out column names and values
while( rs.next() ) {
      for( var i = 1; i <= meta.getColumnCount(); i++ ) {
            print( meta.getColumnName( i ) + ": " + rs.getObject( i ) + "\n"
);
      }
      print( "----------\n" );
}
// cleanup
rs.close();
stmt.close();
conn.close();
```

- **JavaScript Engines**

SpiderMonkey	Mozilla, open-source, written in C	Interpreter, used in Firefox
Rhino	Mozilla, open-source, written in Java	Compiles JavaScript to Java
V8	Google, open-source	Compiles JavaScript to native code

Client-side JavaScript is almost the same with some functions in the server-side, but some keywords, functions, and key functionalities make server-side Javascript distinct to its client counterpart.

- ## What is Node.js?

Node.js, often referred to as just Node, is a powerful tool that can run JavaScript applications on both the server side as well as the client side. Node.js can be used to write static file servers, Web application frameworks, messaging middleware, and servers for HTML5 multiplayer games. This article is a very elementary introduction to Node.js.
Node.js is a server side JavaScript built on Googles V8 JavaScript engine. It is an open source programming language that was developed by Ryan Dahl in 2009. It allows us to build scalable network applications, and is very fast when compared with other server side programming languages because it is written in C and the non-blocking I/O model. It is currently sponsored by Joynet, a software company specialising in high performance container native infrastructure. Node.js can run on Linux, Sun OS, Mac iOS X and Windows platforms.

  - ### Internal Architecture of Node.js

Node.js can be downloaded from *https://nodejs.org/* and is easy to install. Once installed in your system, you will see two of its main parts: the main node executable and the node package manager (NPM) executable. Node executable is simple, and you will pass the name of the main source file – for example, *node sample.js.* The node executable will interpret the source code and execute it. Once it finishes, it will exit back to the shell. The NPM (node package manager) will add new packages to the node executable. To install new modules, enter the command below:

```
npm install name-of-the-module
```

Modules available:
Assert, Buffer, Child_process, Cluster, Crypto, Dgram, Dns, Domain, events, Fs, Http, Https, etc.

Installation of Node.js

- Download Node.js

  The official Node.js website has installation instructions for Node.js: https://nodejs.org.

Command Line Interface

Node.js files must be initiated in the "Command Line Interface" program of your computer. How to open the command line interface on your computer depends on the operating system. For Windows users, press the start button and look for "Command Prompt", or simply write "cmd" in the search field.

Node.js NPM

NPM is a package manager for Node.js packages, or modules if you like.

www.npmjs.com hosts thousands of free packages to download and use.

The NPM program is installed on your computer when you install Node.js

Download a Package

Downloading a package is very easy.

Open the command line interface and tell NPM to download the package you want.

I want to download a package called "upper-case":

Download "upper-case":

```
C:\Users\Your Name>npm install upper-case
```

Now you have downloaded and installed your first package!

NPM creates a folder named "node_modules", where the package will be placed. All packages you install in the future will be placed in this folder.

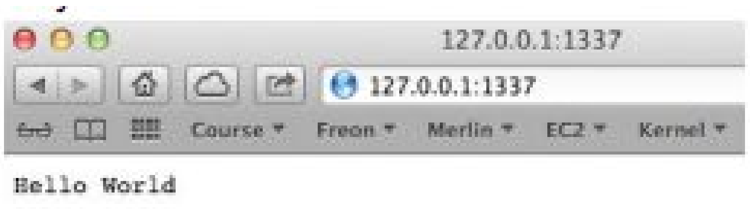
Running HTTP Server in Node.js

Server.js:

```
var http = require('http');
http.createServer(function (req, res) {
      res.writeHead(200, {'Content-Type': 'text/plain'});
      res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

In your brows er: $ node server.js

Run your server:



```
Hello World
```