JavaScript Hit Counters

Dr Jan Pajak

School of Information Technology
Wellington Institute of Technology, Wellington, New Zealand
Jan.Pajak@weltec.ac.nz

This paper presents a summary of basic information regarding hit counters programmed in the JavaScript language and operating on ASP servers. It describes the major functions and capabilities of such hit counters, and explains how these functions can be programmed. It shows source codes, programmed in JavaScript, for a simple hit counter that can be directly utilised on ASP servers. It reveals what other information hit counters can additionally gather, if they are appropriately extended. It also explains JavaScript insertions of code, which are capable of gathering such additional information. In conclusion, the paper discloses to readers main areas of applications of hit counters.

1. INTRODUCTION

In current Internet terminology the name "hit counter" is assigned to a small scripting insertion embedded into web pages and aimed primarily at gathering statistical information regarding visits to a given web site. For example, in standard implementations they count and record a numbers of visitors, or a number of visits. Note that visits to a given web page in Internet jargon are called "hits". Hence the name "hit counter".

This paper aims to **analyse the operation of hit counters and demonstrate their workings by showing the client and server side code of a simple example**. From this the paper proceeds to explain how more complex hit counters typically operate.

To accomplish this aim I will (a) present a JavaScript code for a simple hit counter, (b) explain how the code works, (c) explain how it could be developed further into complex hit counters through including additional capabilities, and (c) share my findings and conclusions regarding such complex hit counters.

2.CODE OF A SIMPLE HIT COUNTER

A typical hit counter is composed of two separate components, which usually are called "client-side" code, and "server-side" code. The **client-side** is always programmed as a very simple piece of code, which is inserted into a given web page. The only purpose of this component is to gather the required information about each visit to a given client web page, and then to send this information to the server-side for further processing. The code of the client-side is always made available to users of hit-counters. However, this code does not say much about how a given hit counter operates and how it is programmed. The **server-side** component of the hit counter is the one, which does all the processing and information gathering. Usually it is located on a distant server, which in many cases is separate from the server on which a given (client) web page is located. Therefore, the actual operation of a given hit counter and the type of information that it gathers is determined by the code contained in this server-side component. But the code of the server-side is NOT made available to anyone. So apart from the team which programmed the code, almost no-one knows what it contains and does. Other parties can only speculate about this code on the basis of information that it produces, and on the basis of data that the client-side sends to it.

In order to give an example of a simple hit counter, presented in Figure 1, a brief "**client**" web page programmed in HTML. This client page contains almost nothing apart from the **client-side code** for a simple hit counter. This code is clearly marked

```
<html>
<head>
<title>A simple hit counter</title>
</head>
<body>
<div align="center"><h1>
<br>Here is a simple hit counter
<br>(refresh if rusty)
<!— Hit counter code begins —>
<br><font color = "green">
<script src="count.js"></script>
<br></font>
<a href=http://Pajak.20m.com><img src="counter.asp" alt="Visit PR655 on:
Pajak.20m.com" width="110" height="40" BORDER="0"></a>
<!— Hit counter code ends —>
</h1></div>
</body>
</html>
```

Figure 1

```
<% @ language = "JavaScript" %>
<%
var current_count = 1;
function Read_Stored_Counter() //Retrieves a stored value
{
var filename="stats.txt"; //This assigns the file name to be read
var file_System_Object = Server.CreateObject("Scripting.FileSystemObject");
var inFile = file_System_Object.OpenTextFile(Server.MapPath(filename));
current_count = parseInt(inFile.readLine());
inFile.Close();
}
function Write_Stored_Counter() //Saves a new value of the hit counter
{
var filename = "stats.txt";
var file_system_object = Server.CreateObject("Scripting.FileSystemObject");
var outfile = file_system_object.CreateTextFile(Server.MapPath(filename));
//The above creates a new file
outfile.WriteLine(current_count);
outfile.Close();
}
function Write_New_Script_File() //Generates a script file named "count.js"
{
var filename = "count.js";
var file_system_object = Server.CreateObject("Scripting.FileSystemObject");
var outfile = file_system_object.CreateTextFile (Server.MapPath(filename));
outfile.WriteLine("var hit_no = " + current_count);
outfile.WriteLine("document.write(hit_no);");
outfile.Close();
}
Read_Stored_Counter();
current_count = current_count + 1;
Write_Stored_Counter();
Write_New_Script_File();
%>
```

Figure 2

with HTML comments, so it is easy to see what it contains.

The essence of the "client-side" code (Figure 1) is to execute the server-side component of the hit counter. This execution is done through running a server-side component named "**counter.asp**" and coded in a scripting language "JavaScript". To simplify the explanation, it is assumed here that this server-side code is located on the same computer as a given client-side web page. But in reality it can be located on a distant server. Then instead of name "counter.asp" of the server-side code, the complete URL of this code needs to be provided. (Further explanations of this code can be found on web sites indicated in the reference section of this paper.)

Figure 2 gives an example of code for a server-side ASP component of the hit counter named "**counter.asp**", which is to be executed each time the above client web page is opened. Note that in order for this code to work, it must be run on a computer that has ASP server capabilities.

If we analyse the ASP code named "counter.asp" and shown in Figure 2, it turns out that it carries out three actions. Namely it:

1.Opens a disk file named "**stats.txt**" and reads data contained in it. In our case this data is represented by one integer number, e.g. 284, which tells how many times a given client web page was hit so-far. But instead of this one number advanced hit counters may use a large database, which contains various data regarding the client web page and/or identities of visitors to this page.

2.Updates the data stored previously in "stats.txt" and writes this updated data back to the "stats.txt" file. In our case this updating depends on incrementing the current count of hits (e.g. from 284 into 285). But in more advanced hit counters it may update an entire database being gathered. So after this stage of processing is completed, the "stats.txt" contains updated values of statistics being gathered.

3.Creates a script file named "count.js". It contains the graphical image of the current count of hits (i.e. 285). This script file is later displayed by the line <script src = "count.js"> </script> from the client-side component of the hit counter. In this way users see the count of hits, which represents the image created from this "count.js" file.

In our example the content of the "**count.js**" file is very simple. It just contains two lines of JavaScript code, e.g.:

```
var hit_no=285;
document.write(hit_no);
```

If these two lines of code are called from a HTML web page through the following tag:

```
<script src="count.js"></script>
```

the effect is to write 285 into a current window. (In our case with the "green" colour, as illustrated on the web site http://free.7host03.com/Pajak.)

Of course, it is also possible to have more elaborate and colourful digits displayed by the hit counter. To obtain such **graphically advanced** hit counters, we just need to prepare a series of pictures of individual digits, putting these digits into appropriately named files (for example: "0.gif", "1.gif", "2.gif", etc.). Then we only need to generate the "count.js" file, which displays the appropriate combination of these digits. An example of the code contained in the "count.js" file, which is to display such "designer" digits, is (for three digits that form a hit_no = 285):

```
document.write('<img src = "2.gif"
border=0> <img src = "8.gif" bor-
der=0> <img src = "5.gif"
border=0><br>');
```

In order to generate such graphically advanced codes of the "count.js" file, the function "Write_New_Script_File()" from the "counter.asp" web page, needs to use a different and more complex algorithm. This new algorithm needs to start from splitting the variable "current_count" from the "counter.asp" into individual digits. Then it should graphically write to the "count.js" addresses of images of these individual digits. An example of this algorithm is provided on web sites (Pajak, 2004).

3. COMPLEX HIT COUNTERS

Ready-made hit counters with basic capabilities, similar to the one explained above, are offered by the majority of Internet space providers. In such implementations these counters usually gather only statistical information. However, various software providers and individual programmers extend these capabilities further, creating very elaborate and very complex hit counters.

The extension of capabilities of hit counters is accomplished through firstly sending additional data to the server-side component of the counter, and secondly through extending the type of information stored in the "stats.txt" database. JavaScript is a powerful language and allows both such extensions to be made quite easily. These more advanced hit counters, which include such complex extensions of the server-side code, can additionally record a whole array of data, both about visits to the client web page, and about identities of visitors. The data may include: date and time of each visit, pages which visitors opened, time spent by these visitors on subsequent web pages, and more. In addition to this, tracking details of visitors can be gathered. For example, complex hit counters are able to record URLs of subsequent visitors, countries in which these visitors are registered, and software these visitors use. In special implementations such counters may even gather information about individual illustrations which these visitors opened and analysed, specific materials these visitors downloaded, features of a given web page in which these visitors were scanning and thus especially interested, etc. Thus hit counters can also be camouflaged security software embedded into web pages, which provide feedback regarding not only visits, but also visitors to given web site. As such, in the majority of cases, code and operation of hit counters is therefore reluctantly revealed to anyone. It is then rather difficult to find comprehensive information about their operation, not to mention finding source code for multifunctional counters.

Such complex, multipurpose hit counters, which offer a multitude of additional information about visitors to a given web page, can be purchased from various specialised software companies. However, while making these counters available, their providers offer them in such a form, that it is only possible to use them, but is almost impossible to learn details of their operation. This means that only code for the client-side component is revealed, while from the server-side only the product of the work is made available. Therefore, in spite of this wide availability of ready-made complex hit counters, it is very difficult, if not almost impossible, to get hold of their source code and descriptions of how these counters actually work.

Apart from advantages resulting from huge data gathering potentials, hit counters have also various limitations. An example of these could be the ambiguity that result from differences between interpretations of "visits" and "hits". Often web site owners may want to know actual "customer visits", as opposed to "hits". After all, "hits" might be incurred each time visitors refresh a page, or even each time they click on a graphics from a given page. Thus totals of "hits" may be slightly misleading, as they do not fully reflect customer activity. Some complex hit counters try to rectify this problem by counting instances of different "URLs" (IPs) that visit a given web page in a given day, rather than counting "hits". In this way, the total count of "visiting URLs" is closer to the real number of visits, although it also does not account for visitors so genuinely interested in a given web page that they visit it more than once in a given day.

4. FINDINGS AND CONCLUSIONS OF THIS PAPER

This paper analyses the operation of hit counters and demonstrates their workings by showing the client and server side code of a simple example. It also explains how hit counters operate as feedback gathering tools. By disclosing this, the paper realises commercial potentials of hit counters. For example, these counters have the capabilities of powerful security software. They are irreplaceable in cases of market research. They turn out to be extremely useful in implementations of market-driven e-commerce. They can also be useful in tracking web intrusions or computer hackers. Moreover, it is important to be aware of their capabilities when we visit various Internet web sites.

REFERENCES

Pajak J. (2004) Unofficial lecture notes (2004-now). Web sites <http://pajak.ownsthis.com/IT6212.htm> and <http://pajak.orcon.net.nz/IT6212.htm>. Accessed April 2004.