

# Analysing IIS Logfiles

Is there anybody out there (visiting our websites)?

by Bob Swart

I've been a webmaster for a long time, and I know that there is nothing more gratifying than knowing that a lot of people are watching and reading your work on the web. For the Developers Group website, my webmastering tasks mainly consist of uploading PDF files, prepared by Joanna. Apart from the DG Magazine (including this very issue), a set of PDF files we've been working on are the O'Reilly newsletters, available at <http://www.richplum.co.uk/oreilly>.

Putting stuff on the web is nice, but we were really wondering if anyone was actually visiting the website, reading what we put on there. Fortunately, being hosted by TDMWeb (excellent service by the way), our web site directory on the server contains a Logfiles directory that contains the IIS logfiles – saved on a per-day basis. So, what better way to find out which parts of the web site are visited (and which not) than to examine these logfiles? Of course, I didn't use Notepad for this, but decided to write a nice little Delphi application for this purpose, which is described in this article.

## IIS Logfiles

The first thing I had to do was to download the IIS logfiles from the TDMWeb server (one for each day), and look inside them for interesting details. For a first test, this resulted in a few dozen logfiles which all start with the following four lines:

---

```
#Software: Microsoft Internet Information Services 6.0
#Version: 1.0
#Date: 2006-04-01 00:01:00

#Fields: date time cs-method cs-uri-stem cs-uri-query cs-username c-ip cs-version cs(User-
Agent) cs(Referer) sc-status sc-bytes
```

---

### Listing 1. Logfile contents

This is very helpful information, especially the fourth line, since it defines the fields that were logged. Not everything might be interesting to analyse, but since I want to build a semi-automatic logfile analysis tool, knowing the names of the fields is a real time saver.

A few dozen logfiles for the Richplum website take up several megabytes already, and aren't really suited to display quickly on screen. So as first step, I wanted to extract some useful information from the logfiles, and place those in a simple Paradox table to start with.

To kick off, I only want to look at the cs-uri-stem, which contains the URL that was requested. This will be the full URL, including the folder in which the requested document was found, and of course the extension. Both the folder and the extension are separate entities of information that I'd like to analyse (in order to determine which folders are the most popular, and which folders are hardly visited at all, as well as being able to filter out certain file types when looking at the results).

The first pass consists of allowing the user to specify the directory that contains the logfiles, and then parsing each logfile in order to extract the cs-uri-stem field, store it somewhere and to determine the maximum length of the URL (I don't want to waste space). As temporary storage space, I decided to use a simple TStringList. At least that would ensure that similar URLs would not end up twice in the list (a huge space saver). Of course, I would have to include the number of times a URL was visited. For that, we can use the fact that a TStringList can contain strings as well as an associated object at the same index value. So for a string at index *i*, we can also have a related object in that index. That's exactly what I need.

As object, I would like to use a special class, which not only counts the number of times this URL was reported in the logfiles, but also lists the unique IP addresses that were used to visit the URL. The IP address can be found in the c-ip field, and is generally considered to be a unique identifier of each visitor (especially now that most people have a permanent connection to the internet using ADSL or Cable). An IP address typically "sticks" for a certain period of time to a certain user, so if the same IP address visits a page twice on a day (or even week), we could consider it the same visitor, and not another visit. The truth (i.e. the actual number of unique visitors) will be somewhere between the number of hits and the number of unique IPs, so if we maintain both these figures, we can make a good guess as to how many times this URL was visited.

The definition for the class that I want to use as associated object in the TStringList is as follows:

---

```
type
    TCounter = class
        visits: Integer;
        IPS: TStringList;
        constructor Create(const IP: String);
    end;

constructor TCounter.Create(const IP: String);
begin
    visits := 1;
    IPS := TStringList.Create;
    IPS.Sorted := True;
    IPS.Duplicates := dupIgnore;
    IPS.Add(IP)
end;
```

---

### Listing 2. TCounter

Note that I place the IP address in a TStringList of its own, setting Sorted to True (which allows for quick insertion), and the Duplicate property to dupIgnore, so we will only include each IP-address just once. That way, we can simply ask the IPS field how many *unique* visitors (i.e. with unique IP-addresses) have visited this specific URL.

## AnalyseLogFile

The next step is the AnalyseLogFile method, which takes a filename as argument and builds the list of URLs. This list is stored in the URL field of type TStringList. In the AnalyseLogFile, we can skip the first four lines of each logfile (as mentioned before), and for each subsequent line skip the first 24 characters that hold the date, time and method. The URL is the next field, followed by a space (or in fact two empty fields for cs-uri-query and cs-username, so we can search for ' - - ' to determine the end of the URL.

Once we have the full URL string, we need to filter out the garbage. By garbage, I mean anything that comes after a ? or & character, and anything within an accidental < or > character. Also, when the URL contains .com, .co.uk or .edu instead of a local path, then we should ignore it as well. Finally, a URL with no dot or with a @ character inside is invalid as well. These checks were included to ensure that 99% of the invalid URLs are ignored.

The URL is then used to extract the folder as well as the extension, since I want to maintain and analyse all three. This can enable us later to filter our the GIF or JPG files for example, when all I want to look at are content pages with the HTM, ASP or even the PDF extension.

When the URL is cleaned, it can be added to the list. If it already exists, we only need to add the IP address (which will be ignored if it already existed). The IP address can be found as seventh field in each line, usually right after the ' - - ' that follows the URL itself, so that's easy to find.

---

```
procedure TMainForm.AnalyseLogFile(const FileName: String);
var
    F: TextFile;
    Str,Str2,IP: String;
    i: Integer;
begin
    AssignFile(F, FileName);
    Reset(F);
    readln(F);
    readln(F);
    readln(F);
    readln(F,Str); // line with field definitions
    while not eof(F) do
        begin
            readln(f,Str);
            Delete(Str,1,24); // skip date, time and method fields
            i := Pos(' - - ',Str);
            if i > 0 then
```

```

begin
  IP := Copy(Str,i+5,16);
  Delete(IP,Pos(#32,IP),16)
end;
Delete(Str,Pos(#32,Str),Length(Str)); // remove other fields
// filter URL field
if Pos('? ',Str) > 0 then Delete(Str,Pos('? ',Str),Length(Str));
if Pos('& ',Str) > 0 then Delete(Str,Pos('& ',Str),Length(Str));
if Pos('< ',Str) > 0 then Delete(Str,Pos('< ',Str),Length(Str));
if Pos('> ',Str) > 0 then Delete(Str,Pos('> ',Str),Length(Str));
if (Length(Str) > 1) and ((Str[1] <> '/') or (Str[2] = '&') or (Str[2] = '_')) then
  Str := '';
if (Pos('.com',Str) > 1) or (Pos('.co.uk',Str) > 1) or (Pos('.edu',Str) > 1) then
  Str := '';
if Pos('.',Str) = 0 then Str := '';
if Pos('@',Str) > 0 then Str := '';
if Pos('stachiewicz',Str) > 0 then Str := '';
if (Length(Str) > 1) and (Str[Length(Str)] = '.') then
  Delete(Str,Length(Str),1);
// still anything left?
if Str <> '' then
begin
  i := URL.IndexOf(Str);
  if i >= 0 then // already found
  begin
    with URL.Objects[i] as TCounter do
    begin
      Inc(visits);
      IPS.Add(IP)
    end
  end
else
begin
  Str2 := Str;
  while Pos('/',Str2) > 0 do Str2[Pos('/',Str2)] := '\';
  if FileExists('..\html' + Str2) then URL.AddObject(Str,TCounter.Create(IP))
end
end
end;
CloseFile(F)
end;

```

---

### Listing 3. AnalyzeLogFile

## Create LogTable

Once all logfiles are parsed and we have a full URL TStringList, we can traverse it once more to determine the maximum length of the directory, full URL and extension fields. With that information in mind, we can finally create a new logfile table and dump the contents of the URTL TStringList in it, as shown in the CreateLogTable method.

I'm starting with a local Paradox table, so can use the TTable component to set the TableType, TableName and then add FieldDefs information for the URL, Dir, Ext, hits and IPs fields. The URL field will be unique, but there will be multiple URLs sharing the same Dir, and the same applies to the value of the Ext column. The hits and IPs columns are simply integer fields that contain the actual information we're looking for. Note that I'm optionally adding an index for the Dir and Ext fields, something that might come in handy later if we want to search or sort by the Dir or Ext fields.

---

```

procedure TMainForm.CreateLogTable;
var
  i,j: Integer;
  UrlLen, DirLen, ExtLen: Integer;
begin
  UrlLen := 0;
  DirLen := 0;
  ExtLen := 0;
  for i:=0 to URL.Count-1 do
  begin

```

```

    if Length(URL[i]) > UrlLen then UrlLen := Length(URL[i]);
    j := PosEx('/',URL[i],2);
    if j > DirLen then DirLen := j;
    j := Length(URL[i]);
    while (j > 0) and (URL[i][j] <> '.') do Dec(j);
    if j > 0 then
        if (Length(URL[i]) - j) > ExtLen then ExtLen := Length(URL[i]) - j;
    end;
// create logfile table
tblLogFile.TableType := ttParadox;
tblLogFile.TableName := 'Logfiles.db';
tblLogFile.FieldDefs.Add('URL', ftString, UrlLen, False);
tblLogFile.FieldDefs.Add('Dir', ftString, DirLen, False);
tblLogFile.FieldDefs.Add('Ext', ftString, ExtLen, False);
tblLogFile.FieldDefs.Add('hits', ftInteger, 0, False);
tblLogFile.FieldDefs.Add('IPs', ftInteger, 0, False);
tblLogFile.IndexDefs.Add('', 'URL', [ixPrimary,ixUnique]);
tblLogFile.CreateTable;
//tblLogFile.AddIndex('Dir','Dir',[]);
//tblLogFile.AddIndex('Ext','Ext',[]);
// add logfile into to table
tblLogFile.Active := True;
for i:=0 to URL.Count-1 do
begin
    tblLogFile.Append;
    try
        tblLogFile.FieldName('URL').AsString := URL[i];
        j := PosEx('/',URL[i],2);
        if j > 0 then
            tblLogFile.FieldName('Dir').AsString := LowerCase(Copy(URL[i],2,j-2));
            if tblLogFile.FieldName('Dir').AsString = '' then
                tblLogFile.FieldName('Dir').AsString := '/';
            j := Length(URL[i]);
            while (j > 0) and (URL[i][j] <> '.') do Dec(j);
            if j > 0 then
                tblLogFile.FieldName('Ext').AsString := Copy(URL[i],j+1,Length(URL[i]));
            tblLogFile.FieldName('hits').AsInteger := (URL.Objects[i] as TCounter).visits;
            tblLogFile.FieldName('IPs').AsInteger := (URL.Objects[i] as TCounter).IPS.Count;
            tblLogFile.Post;
        except
            tblLogFile.Cancel
        end
    end
end
end;

```

---

#### Listing 4. CreateLogTable

Finally, once the table is created, it's a simple matter of traversing the URL TStringList again, and for each item in the TStringList, append a new record to the table with the values for the five fields.

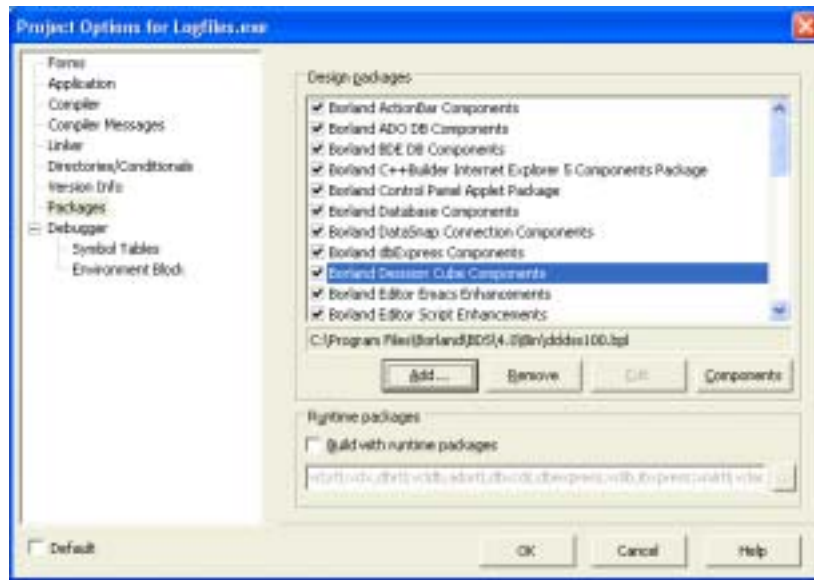
Once we have everything in a Paradox table, we can run some SQL queries against it, for example selecting the number of hits and IPs for the given Ext and Dir fields (ignoring the individual URLs, but just telling us how "busy" the directories have been, and which file types have been serviced by our web server.

```
SQL: SELECT Ext, Dir, Sum(hits), Sum(IPs) FROM Logfiles GROUP BY Ext, Dir
```

The output of such a SQL query can be presented in a grid, but it would be more helpful if we could use a slightly more flexible component that would allow us to expand dimensions (the Ext and Dir fields), and swap summaries (the Sum(hits) and Sum(IPs) values). This is actually exactly what the TDecisionCube – or actually the TDecisionGrid – component is good for.

### Decision Cube

Today, I'm using Delphi 2006, and while previous versions of Delphi Enterprise contained a nice set of components under the label Decision Cube, these are not installed by default in the Delphi 2006 tool palette. Not by default, but it's still included. All we have to do is install the correct design-time package dclcss100.bpl which can be found in the BDS\4.0\Bin directory (see also figure 1).



**Figure 1. Project Options**

With this package installed, the Decision Cube components are all available. The SQL query I defined earlier will be feeding the TDecisionCube component, which in turn feeds the specific Decision Cube visual controls like the DecisionGrid, DecisionPivot and DecisionChart. We won't be using the last of these, but the other two will come in quite handy. To connect the DecisionCube with the visual controls, we need a DecisionSource component in between. Just like working with regular data-aware controls, you only need to set a few properties to tie everything together.

The DecisionPivot is used to define the dimensions and summaries that are displayed in the DecisionGrid. There are typically two sets of dimensions: one at the left side (the y-axis) and one at the top side (the x-axis). Each axis can have zero or more dimensions open at the same time, with the summaries shown as data cells. For our example SQL statement, we have two dimensions: Ext and Dir. We could have added the URL itself as dimension as well, but this would lead to a big explosion of data (each additional dimension increases the amount of required memory by the number of possible values in the dimension, so it's not advisable to use dimensions with a few hundred possible values).

## DecisionGrid

Having multiple summaries allows you to switch from one set of values to another. In our case, the Sum(hits) and Sum(IPs) are our summaries, so we can toggle between viewing all visits (the hits) or just the unique visitors (the IPs). Figure 2 shows the hits in a DecisionGrid of Dir vs. Ext. And specifically, I've focussed on the PDF files in the oreilly directory, to see if these O'Reilly PDF newsletters are being read at all on our web site.

	SUM OF hits												
	Dir						Ext						
	asp	css	gif	png	txt	xml	asp	css	gif	png	txt	xml	Sum
/	11343		5468									36581	22390
about	1287												1287
books	8128		484	3143									31748
contact	1342												1342
ftp	3041												3041
id	1482												1482
downloads	2843						422	1217		4432		2862	30886
images			24987	802					3334				38913
ip	1914												1914
ipd				8									8
links	6207												6207
msg								3629					3629
magazine	3646		3088					11925					14659
meetings	1447							4988					6435
news	1251												1251
offers								302					302
orff	872							4971					5843
services	428												428
support	841												841
topics	84	83											167
Sum	42247	83	5468	36589	5751	422	26582	3334	4432	36581	2862	143408	

**Figure 2. DecisionGrid and DecisionPivot**

Well, from the looks of it, there are some people reading it. Some directories are visited more often, like the meetings and magazine directory (would be a shame if it wasn't).

## Normal Grid

Now that we've seen that the entire directory is indeed visited, I would like to see some more details, like which individual URLs (the actual newsletters) have been read, and how many times. For this, we simply need to open the table in a "normal" Grid, but as a little help I'm adding a drop-down combobox on top of the TDBGrid to allow the user to select a directory. For this, we must ensure that all available directories are listed in the drop-down combobox, which is done in the FormCreate as follows:

```
procedure TVisitForm.FormCreate(Sender: TObject);
begin
  cdsLogCube.Active := True;
  cdsLogFile.Active := True;
  cbDir.Sorted := True;
  cdsLogFile.First;
  while not cdsLogFile.Eof do
  begin
    if Length(cdsLogFile.FieldByName('Dir').AsString) > 1 then
      if cbDir.Items.IndexOf(cdsLogFile.FieldByName('Dir').AsString) < 0 then
        cbDir.Items.Add(cdsLogFile.FieldByName('Dir').AsString);
    cdsLogFile.Next;
  end;
  cdsLogFile.First;
end;
```

### Listing 5. FormCreate

Selecting a specific directory from the drop-down combobox will assign this directory as filter to the table, as shown in the cbDirChange event handler:

```
procedure TVisitForm.cbDirChange(Sender: TObject);
begin
  cdsLogFile.Filtered := False;
  if cbDir.ItemIndex > 0 then
    cdsLogFile.Filter := 'Dir = \'' + cbDir.Items[cbDir.ItemIndex] + '\'';
  else
    cdsLogFile.Filter := '';
  cdsLogFile.Filtered := cdsLogFile.Filter <> '';
end;
```

### Listing 6. cbDirChange

As an example, let's select the meetings directory, and see which meeting PDF files have been read in the previous period. This can be seen in Figure 3.

URL	Directory	Ext	Number of Hits	Percent	Average Number of Hits Per IP (2006)
meetings/20050830.pdf	meetings	pdf	11	0	1.11
meetings/20050815.pdf	meetings	pdf	62	23	2.76
meetings/20051217.pdf	meetings	pdf	84	28	3.36
meetings/20051118.pdf	meetings	pdf	92	32	2.88
meetings/20060117.pdf	meetings	pdf	226	131	1.82
meetings/20060211.pdf	meetings	pdf	808	278	1.87
meetings/20060218.pdf	meetings	pdf	484	238	2.08
meetings/20060314.pdf	meetings	pdf	670	307	2.18
meetings/20060413.pdf	meetings	pdf	896	381	2.34
meetings/20060609.pdf	meetings	pdf	408	218	2.03
meetings/20060612.pdf	meetings	pdf	260	127	2.06
meetings/20060711.pdf	meetings	pdf	324	142	2.28
meetings/20060811.pdf	meetings	pdf	218	127	2.08
meetings/20060817.pdf	meetings	pdf	243	146	2.14
meetings/20061113.pdf	meetings	pdf	318	143	2.21
meetings/index.asp	meetings	asp	2447	108	2.49

Figure 3. Meetings

As you can see, the number of hits and the unique IPs are not equal. So sometimes, a user (someone sharing the same IP-address) has visited or downloaded the same PDF file. Which might happen if you first see the agenda for a meeting, and then later decide to go but want to check the schedule again.

To identify the ratio of number of hits and the number of unique IPs, I've defined a new calculated field, visible in the normal Grid with the "Average Number of Hits Per IP Visit" caption. This calculated field is of course implemented as follows:

---

```
procedure TVisitForm.cdsLogFileCalcFields(DataSet: TDataSet);
begin
    DataSet.FieldByName('HitsPerIP').AsFloat :=
        DataSet.FieldByName('Hits').AsInteger / DataSet.FieldByName('IPs').AsInteger;
end;
```

---

### Listing 7. OnCalcFields

## Sorting TDBGrid

What you may also have noticed from Figure 3 is the fact that the first caption has a slightly different colour. This is a feature to identify by which column I've sorted the resulting dataset. By default you may want to have it sorted on the URL, but sometimes you may want to sort it by Number of hits, Unique IPs or just by Directory or Ext. Note that you cannot sort it by the last field (the Average Number of Hits Per IP Visit), since that's a calculated field.

To sort on a different column, just click on the column header in the TDBGrid. This behaviour can be implemented by writing a little code in the OnTitleClick event handler, setting the IndexFieldName of the dataset. Note that this won't work with all kinds of DataSets, but it works just fine with a TClientDataSet. And in fact, I've loaded the contents of the logfile table inside an in-memory TClientDataSet to speed up the application, and make it independent of the BDE (or any other "real" database for that matter).

---

```
procedure TVisitForm.DBGrid1TitleClick(Column: TColumn);
var
    i: Integer;
begin
    for i:=0 to DBGrid1.Columns.Count-2 do
        DBGrid1.Columns[i].Title.Color := clBtnFace;
    if Column.FieldName <> 'HitsPerIP' then
        begin
            cdsLogFile.IndexFieldNames := Column.FieldName;
            Column.Title.Color := clSkyBlue;
        end
    else cdsLogFile.IndexFieldNames := '';
end;
```

---

### Listing 8. OnTitleClick

And even the result of the SQL command shown earlier, to feed the DecisionCube, has been saved to disk and loaded into a TClientDataSet component. As long as you keep the TClientDataSet components "active" at design-time (with the contents in memory, stored in the DFM file), without a value assigned to their filename, you can actually produce a single stand-alone executable that contains the data itself as well. Make sure to add the MidasLib unit to the uses clause of your project, so you even do not have to deploy the MIDAS.DLL with the application. The only "disadvantage" is that you actually have to compile and re-deploy the stand-alone executable once the data changes (i.e. when you've imported and analysed a new set of logfiles). But for clients and elephant distribution purposes, this is really easy.

Anyway, Figure 4 shows another example, of the magazine directory, sorted by extension. Note that this list does not contain the regular password-protected magazine issues, but rather the "old" 2004 issues which can be accessed freely by anyone. I'm actually amazed by the fact that we've had several thousand downloads or views (even if the unique IPs indeed mean "only" a few hundred visits in the short timespan I've been analysing the logfiles). If a non-member one day reads this article, please consider joining – this is a nice group, and you won't be sorry (shameless plug)...

URL	Directory	File	Number of Hits	Unique Hits	Average Number of Hits Per Hit
magazine/index.asp	magazine	asp	1046	627	1.68
magazine/DIG-FOOT.gif	magazine	gif	362	362	1.18
magazine/DIG-HEAD.gif	magazine	gif	370	367	1.21
magazine/DIG-REDLINE.gif	magazine	gif	356	360	1.19
magazine/dg200401.pdf	magazine	pdf	2257	271	7.59
magazine/dg200402.pdf	magazine	pdf	1583	240	6.40
magazine/dg200403.pdf	magazine	pdf	1281	233	5.52
magazine/dg200404.pdf	magazine	pdf	4382	830	7.88
magazine/dg200405.pdf	magazine	pdf	1822	239	6.84
magazine/dg200406.pdf	magazine	pdf	1838	286	6.00

**Figure 4. Magazine**

Is this it? Almost. There are some ideas and new wishes. It would be nice to have an idea of the period in which URLs were viewed. That can be combined with the DecisionGrid as well, with either the month or the weeknumber as additional an dimension to work with.

Finally, just as I finished writing this article, I discovered an even better way to filter out invalid URLs from the logfiles: the cs-Status field contains the 200 OK code, or an error code in case the page wasn't retrieved correctly. If we simply ignore all non-200 status lines, then this will simplify the filter for invalid URLs significantly. That's left as an exercise for the reader. If you want the source code for the project, take a look at our downloads page (it has been visited a few times before, and we can even publish a top 10 of downloaded files if people are interested).

## Summary

In this article, I've described a 2-pass way to analyse IIS logfiles, put an extract of their information first in a TStringList and then in a Paradox table (and later a TClientDataSet in-memory table). Delphi specific techniques include using the associated object with a string in a TStringList, using the DecisionCube and related components, and filtering and sorting the data in a TDBGrid.



*Bob Swart [b.swart@chello.nl](mailto:b.swart@chello.nl) (aka Dr Bob - [www.drbob42.net](http://www.drbob42.net)) is a software developer, author, trainer, consultant and webmaster for his own one-man company, Bob Swart Training & Consultancy in Helmond, The Netherlands. He writes for numerous computing magazines as well as his own training material, and is also webmaster to the Developers Group.*