# Chapter 4

# Examining Network Traffic and Performance Logs with Log Parser

**Scripts and Samples in this Chapter:**

- **Getting into the Netmon Format**
- **Getting Started with Log Parser's Netmon Format**
- **Finding Soft Errors in TCP Requests**
- **Log Parser, Netmon and Proxy Servers**
- **Using Netmon and Log Parser to Watch for Worms/Intruders**
- **Basic NT Performance Log Queries**
- **Advanced NT Performance Log Queries**

# In This Toolbox

Network capture files recorded by Microsoft Network Monitor (Netmon) are a great low-level way to understand what is happening inside a network application or on a network. While the user interface of this application is generally very useful, it struggles with complicated data search patterns and in understanding the relationships between multiple data packets.

Log Parser quite nicely fills this gap by giving you direct access to all the raw data in the packet and puts the SQL query capability of Log Parser to use in finding what can often be a needle in a haystack for a tester or network administrator. Additionally, in this toolbox you will learn how to take this same concept and apply it to NT performance log data and also make it more manageable and more useful.

# Reading Netmon Capture Files with Log Parser

Like other types of large data sources, network trace data can be gigantic in size and its very nature can make it look very arcane, even to a well-trained eye. Even with that in mind, Log Parser can be used to reveal networking data that has previously been either unavailable to an administrator or difficult to retrieve at best. However, this data is easily reachable with Log Parser.

For example, do you run a website with an ISAPI DLL (Internet Server Application Programming Interface Dynamic Link Library) that exposes public, user-specified function calls? Do you want to reproduce the parameters that are generating a page that is not classified by IIS (Internet Information Server) as an "error," but is still not generating the expected response intended for the user? You only know what external URL (Uniform Resource Locator) the user sees and not what the actual request was since it's a server to server call and the end user cannot see the actual URL. The other server is external and you do not have access to their server's logs, so you can't look there for the referring URL. You're stuck, right? Wrong. Network captures can tell you the URL that the client asked for where an IIS/HTTP (Hypertext Transfer Protocol) log cannot, since it technically is not an error to IIS. This is often referred to by administrators as a *soft* error, which is basically defined as a suboptimal result that is not severe enough to be labeled an exception by the application. Finding soft errors is where using network captures can shine since all the raw data is available to the investigator. Network captures offer raw access to TCP (Transmission Control Protocol) packet information and Log Parser presents it in straightforward fields without missing or arcane descriptions, also providing an optional, connection-oriented view that shows all the packets in a particular connection. This connection includes all the packets in a request stream from start to finish and can be accessed by using the **-fmode:TCPCONN** switch to Log Parser.

**NOTE**

While there are many other network capturing tools available in the enterprise space, Log Parser currently supports the .cap file format used by Microsoft Network Monitor only.

# Getting into the Netmon Format

As mentioned previously, Log Parser plainly identifies all the fields inside the TCP packet, which allows the administrator to spend his or her time investigating and not deciphering the packet's format and fields. See Tables 4.1 and 4.2 for details on all the network fields that are exposed by Log Parser's Netmon input format.

Also present in Log Parser's Netmon format are the following optional switches:

- **-fMode** TCPIP|TCPConn

```
Field mode: TCPIP: each record is a single TCP/IP packet;

TCPConn: each record is a single TCP/IP connection
[default value=TCPIP]
```

- **-binaryFormat** ASC|PRINT|HEX

```
Format of binary fields [default value=ASC]
```

**Table 4.1** Netmon Capture File Fields and Properties in Raw Format

| Property | Field | Description |
|---|---|---|
| Frame Number | Frame | Relative Ethernet frame number in the capture file starting from 1. |
| Time of day | DateTime | W3C Timestamp on each frame. |
| Frame size in bytes | FrameBytes | The size of the Ethernet frame, up to a maximum value of 1514 bytes, including all TCP and IP (Internet Protocol) data. |
| Source MAC address | SrcMAC | The source server's Layer 2 network or Media Access Control (MAC) address. |
| Source IP address | SrcIP | The IP address of the host sending the packet. |
| Source TCP port | SrcPort | The TCP socket that the packet is originating from. |
| Destination MAC address | DstMAC | The destination server's Layer 2 network or MAC address. |
| Destination IP address | DstIP | The IP address of the host receiving the packet. |
| Destination TCP port | DstPort | The TCP socket that the packet is bound for. |

Continued

**Table 4.1** Netmon Capture File Fields and Properties in Raw Format

| Property | Field | Description |
|---|---|---|
| IP Protocol Version | IPVersion | IP protocol version—V4 or V6. |
| IP Time to live | TTL | The amount of time in seconds that the packet is allowed to live. In practice, this is used as a maximum hop count for the packet. Every router the packet crosses must decrement this counter by one. Once the counter is zero, the packet must be discarded by the router and the sender notified via ICMP (Internet Control Message Protocol) messages that the TTL (time to live) has expired in transit. This prevents erroneous or otherwise malicious packets from circulating a damaged network indefinitely. |
| TCP flags | TCPFlags | Flags in the TCP header designating the type of packet—URG, ACK, PSH, RST, SYN, and FIN. They are abbreviated in Log Parser by the 1st letter. |
| TCP Sequence number | Seq | The TCP sequence number that was chosen by the original source host to designate the TCP connection. |
| TCP ACK number | Ack | The TCP acknowledgement number that was chosen by the original source host for the destination server to use when responding to the TCP connection request. |
| TCP Window size | WindowSize | The maximum amount of TCP data in bytes that the sending server will allow to be outstanding between the two hosts without the reception of an acknowledgement. |
| TCP data size in bytes | PayloadBytes | The size of the TCP portion of the packet, up to a maximum value of 1460 bytes. |
| TCP data | Payload | The raw TCP data that is being transmitted, in ASCII or HEX format. |
| TCP Connection / session identifier | Connection | A relative connection number that is assigned by Log Parser to every established TCP connection in the capture file. This is done by deriving data from the TCP sequence/Acknowledgement numbers/source and destination ports and other factors. |

**Table 4.2** Netmon Capture File Fields and Properties in Connection Format

| Property | Field | Description |
|---|---|---|
| Starting Frame Number of session | StartFrame | Starting Ethernet frame number in the session. |
| Ending Frame Number of session | EndFrame | Ending Ethernet frame number in the session. |
| Total frames in session | Frames | Total number of individual frames in the session. |
| Source MAC address | SrcMAC | The source server's Layer 2 network or MAC address |
| Source IP address | SrcIP | The IP address of the host sending the packet. |
| Source TCP port | SrcPort | The TCP socket that the packet is originating from. |
| Destination MAC address | DstMAC | The destination server's Layer 2 network or MAC address. |
| Destination IP address | DstIP | The IP address of the host receiving the packet. |
| Destination TCP port | DstPort | The TCP socket that the packet is bound for. |
| Time taken to complete session | TimeTaken | The amount of time, in seconds, that the session took to complete. |
| Time of day | DateTime | W3C Timestamp on each frame. |
| TCP data size in bytes on packet from client | SrcPayloadBytes | The size of the TCP portion of the user's packet, up to a maximum value of 1460 bytes. |
| TCP data included in the client's request | SrcPayload | The end user's request to the server host in ASCII. |
| TCP data size in bytes on packet from server | DstPayloadBytes | The size of the TCP portion of the response packet, up to a maximum value of 1460 bytes. |
| TCP data included in the server's response | DstPayload | The server's response to the user host in ASCII. |

**TIP**

Full Microsoft TCP/IP implementation details can be found at www.microsoft.com/technet/itsolutions/network/deploy/depovg/tcpip2k.mspx.

# Getting Started with Log Parser's Netmon Format

Let's briefly look over two basic queries in the Netmon format to give some context. These are basic IIS requests for web pages. One is shown in connection format (**-fmode:TCPConn**) and the other is shown in the standard TCPIP format, which is the default. Use command syntax like this for any Netmon query in Log Parser:

```
logparser.exe file:Ch04TCPConn.sql -fmode:TCPConn


---Ch04TCPConn.sql---
SELECT
               SrcIP,
               DstIP,
               DstPort,
               SrcPayload,
               DstPayload
FROM      FOO.cap
WHERE     DstPort = 80
OR             SrcPort = 80
AND     DstPayload like '%200%'

Output:

SrcIP           DstIP           DstPort SrcPayload
12.52.84.19    61.4.12.13      80       GET foo.gif HTTP/1.0..Accept: image/gif, image/x-
xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*..Accept-Language: en-
us..Accept-Encoding: gzip, deflate..User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows
98).. X-Forwarded-For: 19.24.114.2..Via: 1.1 TTCache04 (Jaguar/3.0-59)..

DstPayload
HTTP/1.1 302 Object moved..Server: Microsoft-IIS/5.0..Date: Mon, 27 Sep 2004 20:42:34
GMT..P3P:CP="BUS CUR CONo FIN IVDo ONL OUR PHY SAMo TELo"..Location:
http://foo.com/foo.gif..Content-Length: 121.. image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, application/x-shockwave-flash, */*..Cache-control:
private....<head><title>Object moved</title></head>.<body><h1>Object Moved</h1>This
object may be found <a HREF="">here</a>.</body>.
---Ch04TCPConn.sql---
---Ch04TCPIP.sql---
SELECT
               SrcIP,
               DstIP,
               DstPort,
               Payload
FROM      FOO.cap
WHERE     DstPort = 80
```

```
OR      SrcPort = 80

Output:
SrcIP         DstIP          DstPort Payload
19.175.37.8    21.21.12.13    80      GET  foo.gif HTTP/1.0..Accept: image/gif, image/x-
xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*..Accept-Language: en-
us..Accept-Encoding: gzip, deflate..User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows
98).. X-Forwarded-For: 19.214.114.2..Via: 1.1 TTCache04 (Jaguar/3.0-59)..Connection:
Keep-Alive....
---Ch04TCPIP.sql---
```

# Finding Soft Errors in TCP Requests

Now that we have seen a little of how Log Parser displays data, let's move more into some troubleshooting. As mentioned in the introduction to this chapter, network requests using platform technologies like IIS/Java/etc. often defy common definitions of failure. This usually is the result of an application failure (for example, a custom ISAPI) being too granular for a generic host application (like IIS), hence the term *soft* failure. Regardless of the definition or the reason, the resultant problem is that the user is getting the wrong response, and the administrator and the application developer have to find out the root cause. The IIS log doesn't show anything more than a 200 [OK] response to a million or so requests and there is not any way in the IIS log to tie those requests to the responses that the client is seeing. The bad response has to be backtracked to the exact parameters that were passed to the IIS web server so that the problem can be found.

**TIP**

Example: `logparser.exe file:Ch04TCPConn.sql?filename=FOO.cap -fmode:TCPConn`
Notice that the capture filename is passed to the SQL file as a parameter. This technique is reused throughout this chapter.

```
---Ch04ErrantHTTP.sql---
SELECT
        TimeTaken,
        SrcPayload,
        DstPayload
FROM    %filename%
WHERE   DstPayload like '%UseMethodX%'
ORDER   BY SrcPayload
---Ch04ErrantHTTP.sql---
```

This example shows the requests that had a response, which included **UseMethodX** in the ASCII portion of the TCP response packet to the user. While IIS logs will only show the request and the IIS/WIN32 status of the request, this output is organized around the reply that was

received by the client and also shows the full data portion of the server's reply to the request. Here is an example of a reply to this query:

```
2046.875000
GET /redirect.dll?UseMethodX
HTTP/1.1 302 Object moved..Server: Microsoft-IIS/5.0..Date: Mon, 27 Sep 2004 20:42:33
GMT..P3P:CP="BUS CUR CONo FIN IVDo ONL OUR PHY SAMo TELo"..Location:
http://foo.com/OLDredirectpage.htm..Content-Length: 121..Content-Type: text/html..Set-
Cookie: FOOCOOKIE=thiscookievalue; expires=Mon, 04-Oct-2021 19:00:00 GMT;
domain=.foo.com; path=/..Cache-control: private....<head><title>Object
moved</title></head>.<body><h1>Object Moved</h1>This object may be found <a
HREF="">here</a>.</body>.
```

Perhaps you suspect that the error centers around cookies that are not being passed, or are being passed erroneously. This will retrieve the URL requested, the HTTP status code (302), and also all the headers and cookies that were sent back. We can check for cookies being set by the server and/or presented by the client. For server cookies it is:

```
---Ch04Server_Cookies.sql---
SELECT
SUBSTR(EXTRACT_TOKEN(Payload,1,'Cookie:'),0,INDEX_OF (EXTRACT_TOKEN(Payload, 1,
'Cookie:'), '..'))
AS ServerCookie
FROM %filename%
WHERE ServerCookie is not NULL
AND SrcPort = 80

output:
COOKIE1=UM=; expires=Tue, 26-Apr-2022 12:00:00 GMT; domain=.FOO.com; path=/
```

For client cookies it is:

```
SELECT
SUBSTR(EXTRACT_TOKEN(Payload,1,'Cookie:'),0,INDEX_OF (EXTRACT_TOKEN(Payload, 1,
'Cookie:'), '..'))
AS ClientCookie
FROM %filename%
WHERE ClientCookie is not NULL
AND DstPort = 80

output:
XC1=V=3&PGID=a978000eba114d48888576637e3b5729;
```

Master Craftsman

## Understanding TCP Sequencing

Please note, while Log Parser can identify packets that should be grouped together by using the **–fmode:TCPConn** switch, it is useful to know how to sequence TCP packets and connections manually as well, especially when looking at network traffic behind proxies and accelerators that do not always close TCP connections like normal clients. In the proxy cases, using the **–Fmode:TCPConn** switch results in a relatively small number of unique connections with many requests appended together. To isolate specific requests in these cases, you will have to read the sequences manually with Log Parser. You may have to manually examine the network frames a few times (select * from FOO.cap) to understand your connection profiles (how many bytes are on the request/response, etc) before you can write reliable, specific queries, but it is certainly possible and useful with a little practice.

When manually looking at TCP captures, especially in captures from very busy networks, one way to follow a particular request from start to finish is to leverage the TCP sequence and acknowledgement numbers attached to each TCP packet. The following is a basic explanation of that process, but the full explanation of TCP sequencing is present in the TCP RFC, Section 3.3, located here: www.ietf.org/rfc/rfc793.txt

In a simple TCP data request, the client sends a TCP sequence number on any request to a remote server to uniquely identify that packet. This number is chosen on connection startup by the client and is 32 bits in length. The host also sets any applicable TCP flags for the packet, based upon the packet's profile. For example, on an initial request, the SYN (Synchronize) field in the TCP header will be set by the client. On the first server reply, the ACK (Acknowledge) header will be set in addition to the SYN header and so on. When data is appended to a TCP request, the PSH (Push Data) flag is set in addition to the ACK flag. When a client wants to end the connection, it will include the FIN flag in its packet to the server.

Any time that the ACK flag is set in the TCP header, a non-zero acknowledgement number is present in the packet as well. Acknowledgment numbers are generally formed when a host will "Acknowledge" another host's request by sending back the sending host's TCP sequence number in the Acknowledgement field of its response packet to "Acknowledge" the receipt of that specific packet. Additionally, any host "Acknowledging" TCP packets will increment this Acknowledgement number slightly using a few simple guidelines:

- If a client's packet only has the SYN flag present (in the case of the 1st packet in a TCP connection), the Acknowledgement number will be 0, since no data is being "acknowledged." However, the server's reply to this packet will have an Acknowledgement number of the clients initial TCP Sequence number (also known as ISN) + 1. For Example, when the Server acknowledges a packet from the client with ONLY the SYN flag set:

  Server's Acknowledgement number = ISN + 1

Continued

Server's TCP Sequence number = New 32 bit TCP sequence number that it generates.

- If a packet is received, but no data is being sent, the host will acknowledge that packet and increment the Acknowledgment number + 1. For example, when a Server acknowledges a packet from the client with the ACK flag set, but no data:

  Server's Acknowledgement number = Client's TCP Sequence number +1

  Server's TCP Sequence number = Client's ACK number

- If the client OR server is ALSO acknowledging the receipt of TCP data (the packet it is acknowledging had the PSH flag set), it will increment its Acknowledgment number by the amount of TCP data bytes it received. For example, when a Server acknowledges client's data:

  Server's Acknowledgement number = Client's TCP Sequence number + data bytes

  Server's TCP Sequence number = Client's ACK number

# Log Parser, Netmon and Proxy Servers

If a server is behind a proxy, finding the same data as in the previous example might prove difficult since most proxies re-use TCP connections. Consequently, using Log Parser's TCPConn mode may not give us the granular data that is needed. This query, while a bit slow, will give the TCP request that was the predecessor to any response. If what you need did not occur immediately before the reply in the request stream, but rather is one or more frames back in the chain, repeat the following process or further nest the following query to get where you need.

```
--- Ch04ManualTCP.sql---
SELECT
        Frame,
        DateTime ,
        FrameBytes,
        SrcPort,
        DstPort,
        TCPFlags,
        Seq,
        Ack,
        WindowSize,
        PayloadBytes,
        Payload
FROM    %filename%
WHERE   Payload like Payload '%UseMethodX%'
OR
```

```
   (SEQ IN(
SELECT
        Ack from %filename% where Payload like '%UseMethodX%')
)
--- Ch04ManualTCP.sql---
```

## Using Netmon and Log Parser to Watch for Worms/Intruders

While there are a lot of intrusion methods out there on the market, it is still useful to look at your raw network packets to see who is doing what on your network. You will always find something going on that you did not previously know about, benign or otherwise. If you suspect some machines on your network are owned and trying to infect your servers, you might perform a quick network trace and use a query like this one, which could be easily tuned to look for the specific exploit you are worried about. This one looks for machines infected by many of the known worm entry points/exploits on the network and calls out machines that may be compromised.

```
---Ch04owned.sql---
SELECT
        DISTINCT REVERSEDNS(srcip) AS SuspiciousMachineName,
        srcip,
        dstport,
        COUNT(srcip) as SuspiciousConns
FROM    %filename%
WHERE   dstport in
(80;137;445;559;1025;1026;1027;135;1434;2745;2535;3127;3410;5000;5554;6129;27374;65506)
GROUP BY srcip,
        dstport
HAVING  SuspiciousConns > 5

output:

SuspiciousMachineName           SrcIP           DstPort SuspiciousConns
------------------------------- --------------- ------- ---------
Coderedworm.foo.com             24.42.23.91     80      6
joe.com                         23.23.18.67     80      6
bob.hacket.ca                   21.7.16.229     80      6
---Ch04owned.sql---
```

# Deriving Data from NT Performance Logs

Ever have a colleague drop off a 1GB NT performance log and ask you to "take a look" at it to see if you find anything unusual? With Log Parser, you can actually take a quick look at a huge performance log and spot problem vectors much easier. The code in this section uses Log Parser

to identify every individual performance counter in an NT performance log and output to the console the minimum, maximum, and average values for each of the counters that are in the log file. This gives the administrator a good chance of seeing something unusual and a much smaller investigative surface to explore. Once you spot a problem, you can query all the granular data with a standard Log Parser CSV query to get every individual reading for a particular NT performance counter that you think is worthy of investigation.

**NOTE**

The following code assumes your NT performance logs are in CSV output format. The code could be modified to accommodate TSV logs if needed.

# Basic NT Performance Log Queries

Before we query the files, let's take a really quick look at gathering that data easily from the command line — by using the resource kit utility LogMan.exe, we can quickly create an NT Performance log collection and start/stop it. Use the following script (Ch04CreateLog.cmd) and the sample NT counter list which follows this code to create a log to analyze with Log Parser and output a graph of the data. It will create the log collection on the local machine and start collecting data. In the example, we just collect for 2 seconds on the local machine, but the script is easy to modify to suit any time needs and LogMan.exe can create logs on remote machines by adding the –S <servername> switch.

**TIP**

LogMan always inserts the computername into each NT performance counter – make sure you account for this in your queries. In the following example, we pass a parameter designating the servername to Ch04QueryMem.sql like this:

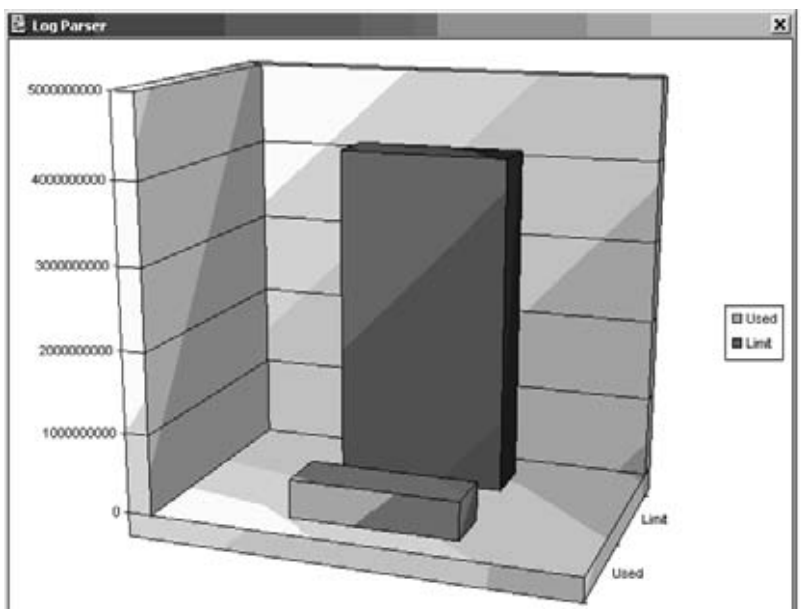
```
logparser.exe file:Ch04QueryMem.sql?filename=memory.csv+server=myServer -view:on -
charttype:Column3d
```

```
---Ch04CreateLog.cmd---
@echo off

if "%1"=="/?" goto :usage
if "%1"=="" goto :usage

REM drop the countergroup if it exists
@echo stopping and deleting any old copies of the logset...
logman stop %1>nul
logman delete %1>nul
REM create a new set
logman create counter %1 -cf %1 -rf 00:00:02 -o c:\perflogs\ -si 00:00:01 -f csv --v
REM start the set for 2 seconds and then stop
```

```
goto :eof
:usage
ECHO Ch04CreateLog.cmd {NTCounterlistFile}
ECHO Example: Ch04CreateLog.cmd Ch04taskman
goto :eof

:eof
@echo on
---Ch04CreateLog.cmd---
---Ch04taskman---
"\Memory\Commit Limit"
"\Memory\Committed Bytes"
---Ch04taskman---


---Ch04QueryMem.sql---
SELECT
        AVG([\\%servername%\Memory\Commit Limit]) as Limit,
        AVG([\\%servername%\Memory\Committed Bytes]) as Used
FROM    %filename%
TO      Memory.gif
---Ch04QueryMem.sql---
```

The output is shown in Figure 4.1.

**Figure 4.1** Output of Ch04QueryMem.sql



# Advanced NT Performance Log Queries

While performance logs are just CSV files, anyone running in an enterprise would be quick to point out that no two NT performance logs are the same—people are always measuring something different with a different number of columns. Astute administrators would be well advised to adapt some of the previous and following code to create canned NT performance log sets and corresponding Log Parser queries. The following section addresses the other side—an NT performance log that has a variable number of columns, which complicates the analysis with Log Parser significantly. However, with the following code, you should be able to handle most NT performance logs without much trouble. The code is commented, so modifying it to adapt to your enterprise or environment should not be difficult.

While Log Parser does not intrinsically support finding column names and then querying for them, you can write add-on code to pull those names and then query for them. The following C# example pulls the column names from any CSV Perfmon file and then stores those names for querying one at a time for minimum, maximum, and average values for the entire file. This can also be accomplished a little more brutally by using a temp file and then interrogating it via a second query inside a batch file. Both examples are given. Which you choose is more of a matter of preference, as the performance is similar.

```
---CH04LP_perfmon.cs---
using System;
using LQC     = Interop.MSUtil.LogQueryClassClass;
using LCI     = Interop.MSUtil.COMCSVInputContextClassClass;

class LogP
{
       public static void Main(string[] Args)
       {
       // construct objects that we just defined in using above

       try
              {
                     LQC obj2 = new LQC();
                     LCI objI = new LCI();

       // getting the number of columns

       objI.headerRow = true;
       string query1 = @"SELECT top1 * from "+Args[0];
       Interop.MSUtil.ILogRecordset recset= obj2.Execute(query1, objI);
       Interop.MSUtil.ILogRecord record = recset.getRecord();

              while (recset.atEnd() != true)
                     {
              for (int I = 4; I < recset.getColumnCount() -1; I++)
```

```
                        {
      //print the NT counternames from the logfile

                                Console.WriteLine(recset.getColumnName(I));

      // Create a SQL query for Log Parser to query the file

string query2 = "select ";
query2 += @"AVG(TO_INT(["+recset.getColumnName(I)+"])) as avg_value_,";
query2 += @"MIN(TO_REAL(["+recset.getColumnName(I)+"])) as min_value, ";
query2 += @"MAX(TO_REAL(["+recset.getColumnName(I)+"])) as Max_value from ";
query2 += Args[0];

//print the counter min/max/avg to the console

Interop.MSUtil.ILogRecordset recset2= obj2.Execute(query2, objI);
Interop.MSUtil.ILogRecord record2 = recset2.getRecord();
Console.WriteLine(record2.getValue(0).ToString()+
" "+record2.getValue(1).ToString()+" "+record2.getValue(2).ToString());
                        }
                    recset.moveNext();
                    }
             }
             catch (System.IndexOutOfRangeException)
      {
                    Console.WriteLine();
                    Console.WriteLine("must specify a valid perfmon logfile to parse");
                    Console.WriteLine("usage: LP_Perfmon.exe <CSV logfile>");
                    Console.WriteLine("example: LP_Perfmon.exe foo.csv");
      }
             catch (System.IO.FileNotFoundException)
      {
                    Console.WriteLine();
                    Console.WriteLine("must specify a valid perfmon logfile to parse");
                    Console.WriteLine("usage: LP_Perfmon.exe <CSV logfile>");
                    Console.WriteLine("example: LP_Perfmon.exe foo.csv");
      }
             catch (System.Runtime.InteropServices.COMException)
      {
                    Console.WriteLine("you must have the Log Parser.dll registered to run
this");
                    Console.WriteLine(@"get it from http://logparser.com");
      }
      }
}
---CH04LP_perfmon.cs---
```

Perform the following steps to use the C# sample (you need the .NET framework installed):

1. Make an Interop wrapper for LogParser.dll.

```
tlbimp LogParser.dll /out:Interop.MSUtil.dll
```

2. Compile the Ch04LP_Perfmon.cs file into an executable (.EXE) with the following command:

```
csc /r:Interop.MSUtil.dll /out:Ch04LP_perfmon.exe  Ch04LP_perfmon.cs
```

3. Run the new executable:

```
Ch04LP_Perfmon.exe <performance log filename.csv>
```

4. Each output line has the NT counter name, the Average Value, The Minimum Value and the Maximum Value on the following line:

```
\\serverName\Memory\Page Faults/sec
743 15.498725 14296.662888
```

5. To drill down on a particular counter that you find interesting:

```
   ---Ch04CounterDrill.sql---

SELECT
[(PDH-CSV 4.0) (Pacific Standard Time)(480)] AS time,
[\\serverName\Memory\Page Faults/sec]
FROM FOO_Log.CSV
ORDER BY time DESC
---Ch04CounterDrill.sql---
```

If you prefer scripting, here is a Windows Shell equivalent.

```
---Ch04LP_perfmon.cmd---
@echo off

if "%1"=="" goto :usage
if "%1"=="-h" goto :usage
if "%1"=="/h" goto :usage
if "%1"=="-?" goto :usage
if "%1"=="/?" goto :usage

REM setup variables
set countername=
set filename=%1
set %filename%=filename

REM make a temp directory and cleanup old ones
```

```
if exist ~lptemp rd ~lptemp /S /Q
md ~lptemp

REM get Column Names onto separate lines
@logparser.exe file:Ch04Top1.sql -q:on -i:csv -o:csv -headerrow:off -headers:off
@logparser.exe file:Ch04CRLF.sql -q:on -i:textline -headers:off -oDQuotes:OFF -o:csv

REM loop through the perfmon file and get the Average/MIN/MAX
FOR /F "skip=3 delims=," %%f in (.\~lptemp\counterlist2.csv) do (
set countername=
set countername=%%f
set %%countername%%=countername
@logparser.exe file:Ch04AVG.sql -q:on -i:csv
@logparser.exe file: Ch04MAX.sql -q:on -i:csv
@logparser.exe file: Ch04MIN.sql -q:on -i:csv)

REM Clean Up
echo cleaning up temp files
rd ~lptemp /S /Q

goto :eof

:usage
@echo Ch04LP_perfmon.cmd (CSVfilename.cmd)
goto :eof
:eof
@echo on
Include files for the previous example:
---Ch04Top1.sql---
SELECT
        TOP 1 *
FROM    %filename%
TO      .\~lptemp\counterlist1.csv
---Ch04LP_perfmon.cmd---
--- Ch04CRLF.sql---
SELECT
        REPLACE_CHR(Text,',','\u000a')
FROM    .\~lptemp\counterlist1.csv
TO      .\~lptemp\counterlist2.csv
--- Ch04CRLF.sql---
--- Ch04AVG.sql---
SELECT
        'Average',
        '%countername%',
        AVG(TO_INT([%countername%]))
FROM    %filename%
--- Ch04AVG.sql---
```

```
---Ch04MAX.sql---
SELECT
        'Maximum',
        '%countername%',
        MAX(TO_REAL([%countername%]))
FROM    %filename%
---Ch04MAX.sql---
---Ch04MIN.sql---
SELECT
        'Minimum',
        '%countername%',
        MIN(TO_REAL([%countername%]))
FROM    %filename%
---Ch04MAX.sql---
```

**TIP**

Windows NT performance counters usually display a very long name with spaces and punctuation included as identifiers. To avoid parsing errors, encapsulate these names in brackets [] as shown in the aforementioned code for Log Parser to correctly identify them as valid column names, for example, `[\\serverName\Memory\Page Faults/sec]`.

# Advanced Graphing Windows NT Performance Data with Log Parser

The Windows NT performance monitor is a good tool for gathering data and generating simple views, but deep analysis of performance logs with multiple data points may prove to be too much for the Perfmon interface, as it mainly supports linear graphs and is designed more for real-time monitoring. Here is a script that will use LogMan and Log Parser to pull virtual memory data for all processes on a system and aggregate it into files that are then graphed to show the data visually.

**TIP**

Log Parser does not support parsing column names as previously mentioned. In order for this example to work, the columns (an the performance counters that they represent) collected per process must be in the same order in each NT performance log file for the columns to line up correctly and show the correct aggregated data. In this case, we only have one data column, so that simplifies the process.

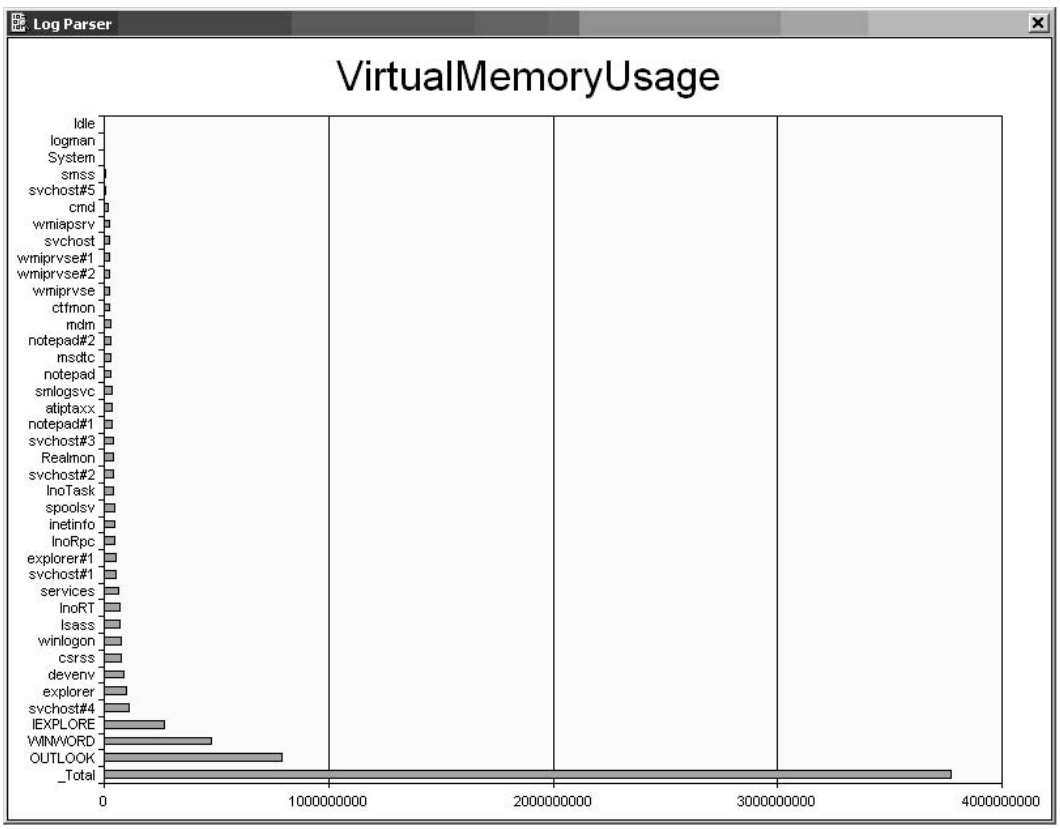
```
---Ch04CreateLog.cmd---
@echo off

if "%1"=="/?" goto :usage
if "%1"=="" goto :usage

REM drop the countergroup if it exists
@echo stopping and deleting any old copies of the logset...
logman stop %1>nul
logman delete %1>nul
REM create a new set
logman create counter %1 -cf %1 -rf 00:00:02 -o c:\perflogs\ -si 00:00:01 -f csv --v
REM start the set for 2 seconds and then stop
goto :eof
:usage
ECHO Ch04CreateLog.cmd {NTCounterlistFile}
ECHO Example: Ch04CreateLog.cmd Ch04taskman
goto :eof

:eof
@echo on
---Ch04CreateLog.cmd---
---Ch04LP_PerProcess.cmd---
@echo off

if "%1"=="" goto :usage
if "%1"=="-h" goto :usage
if "%1"=="/h" goto :usage
if "%1"=="-?" goto :usage
if "%1"=="/?" goto :usage

setlocal ENABLEDELAYEDEXPANSION

REM setup variables

set vALL=
set filename=%1
set %filename%=filename

REM make a temp directory and cleanup old ones
if exist ~lptemp rd ~lptemp /S /Q
md ~lptemp

REM get Column Names onto separate lines
logparser.exe file:Ch04Top1.sql -q:on -i:csv -o:csv -headerrow:off -headers:off
```

```
logparser.exe file:Ch04CRLF.sql -q:on -i:textline -headers:off -oDQuotes:OFF -o:csv
logparser.exe file:Ch04GetProcess.sql -q:on -i:textline  -headers:off -oDQuotes:OFF -o:csv

REM get numbers
REM get the processname
REM and then get the counternames for that process
REM it is best to control the counters with what is gathered -- see logman portion
for /f "delims=," %%w in (.\~lptemp\Processlist.csv) do (logparser -q:on -i:textline
"select Text from .\~lptemp\counterlist2.csv to .\~lptemp\%%w.txt where
SUBSTR(EXTRACT_TOKEN(Text,1,'\Process('),0,INDEX_OF(EXTRACT_TOKEN(Text, 1, '\Process('
),')')) = '%%w'"
REM aggregate the variables together
for /f "delims=;" %%z in (.\~lptemp\%%w.txt) do set vALL=!vALL!Avg^(TO_INT^(^[%%z^]^)^)^,
set vall=!vall:~0,-1!
REM query for the average value in the perflog
logparser -q:on -i:csv -o:csv "select !vALL! from %filename% to .\~lptemp\%%w.csv"
set vall=)

REM cleaning up list files
Del .\~lptemp\counterlist1.csv
Del .\~lptemp\counterlist2.csv
Del .\~lptemp\processlist.csv

REM graph the files
LogParser file:CH04graph.sql -i:csv -headerrow:off -nSkipLines:1 -Charttype:BarClustered -
view:on -GroupSize:800x600

REM Clean Up
REM read out the files we just did – use this if you want to keep
REM the tempfiles below and comment out the RD command
REM @echo files written:
REM @for /f %%q in (.\~lptemp\processlist.csv) do dir .\~lptemp\%%q.csv /b
ECHO cleaning up the rest of the temp files
rd ~lptemp /S /Q
goto :eof

:usage
@echo Ch04LP_PerProcess.cmd (CSVfilename.cmd)
goto :eof

:eof
@echo on
---Ch04LP_PerProcess.cmd---
---Ch04log_vbytes---
"\Process(*)\Virtual Bytes"
---Ch04log_vbytes---
```

```
---Ch04GetProcess.sql---
SELECT
       DISTINCT SUBSTR(EXTRACT_TOKEN(Text, 1, '\Process('),0,INDEX_OF(EXTRACT_TOKEN(Text,
1, '\Process(' ),')'))
FROM  .\~lptemp\counterlist2.csv
TO    .\~lptemp\Processlist.csv
---Ch04GetProcess.sql---
---Ch04Graph.sql---
SELECT
       EXTRACT_TOKEN( EXTRACT_TOKEN(Filename, -1, '\\'), 0, '.') AS ProcessName,
       Field1 AS VirtualMemoryUsage
INTO   allvm.gif
FROM   ~lptemp\*.csv
ORDER BY VirtualMemoryUsage DESC
---Ch04Graph.sql---
```

The output of this is shown in Figure 4.2.

**Figure 4.2** Output of Ch04LP_PerProcess.cmd



**TIP**

This goes without saying, but since I know many of you will chain these scripts together to make this a one command process, here is one caveat – the log collection happens silently: make sure you leave enough time for the logs to finish collecting data before you query them or you might get odd results. You can check to see if it is running or not by using LogMan.exe QUERY (logsetName). You can also use tools like SLEEP.exe to add artificial pauses in NT .cmd scripts between programs.

# Final Touches

The ability to read such low level formats is a testament to the flexible, generic nature of Log Parser. While many other programs and scripts can be used to examine log files, Log Parser's native support of the Netmon format set it apart in many important ways. Ordinary scripts and parsers simply cannot read these files reliably. Where and while Netmon can read it, it cannot read and display the data in the incredibly flexible ways that Log Parser can. Similarly, many programs can read NT performance logs, but not with the unique extensibility and power of Log Parser. Hopefully these Netmon examples will inspire administrators to look more at the low levels of their network to leverage this existing functionality in a much more flexible and simple way than ever before. As for the NT performance logs, hopefully those giant files that are delivered do not look as intimidating as they once did, but rather appear as an opportunity to settle a problem once and for all.