Log Parser

Log parser is a powerful, versatile tool that provides universal query access to text-based data such as log files, XML files and CSV files, as well as key data sources on the Windows® operating system such as the Event Log, the Registry, the file system, and Active Directory®. You tell Log Parser what information you need and how you want it processed. The results of your query can be custom-formatted in text based output, or they can be persisted to more specialty targets like SQL, SYSLOG, or a chart.

The world is your database with Log Parser.

Most software is designed to accomplish a limited number of specific tasks. Log Parser is different... the number of ways it can be used is limited only by the needs and imagination of the user.

If you find a creative way to use it, let us know at <u>www.logparser.com</u>!

Here are some samples to whet your appetite...

Search for Data

Search for the logons of a specific user among the events in the Windows Event Log:

C:\>LogParser "SELECT TimeGenerated, SourceName, EventCategoryName,

Message INTO report.txt FROM Security WHERE EventID = 528 AND SID Appropriate for the second security where events are a second second

<u>File E</u> dit F <u>o</u> rmat <u>V</u> iew <u>H</u> el	P		
TimeGenerated	SourceName	EventCategoryName	Messa 🔼
2004-02-05 11:11:14 2004-06-18 09:56:37 2004-06-19 10:43:10 2004-06-21 08:06:52 2004-06-21 17:50:06 2004-06-22 08:14:05 2004-06-22 09:42:54 2004-06-23 08:49:35 2004-06-24 10:01:06 2004-06-25 07:37:35 TimeGenerated	Security Security Security Security Security Security Security Security Security SourceName	Logon/Logoff Logon/Logoff Logon/Logoff Logon/Logoff Logon/Logoff Logon/Logoff Logon/Logoff Logon/Logoff Logon/Logoff Logon/Logoff EventCategoryName	Succe Succe Succe Succe Succe Succe Succe Succe Succe Succe Messa
2004-06-26 10:13:46 2004-06-26 12:58:55 2004-06-28 08:29:08 2004-06-28 09:43:18 2004-06-30 07:34:23 2004-07-01 07:43:41 2004-07-01 09:26:24 2004-07-02 07:33:53 2004-07-03 19:40:59 2004-07-04 10:41:18	Security Security Security Security Security Security Security Security Security Security	Logon/Logoff Logon/Logoff Logon/Logoff Logon/Logoff Logon/Logoff Logon/Logoff Logon/Logoff Logon/Logoff Logon/Logoff Logon/Logoff	Succe Succe Succe Succe Succe Succe Succe Succe Succe Succe

Create Reports

Create custom-formatted HTML reports:

Report generated by TESTDOMAIN\gabriele on 2003-04-21 23:41:44 <u>Application Security System</u>					
Туре	Time	Source	Category	Event	User
WARN	4/21 15:47:01	Print	None	20	SYSTEM
INFO	4/21 12:05:04	Service Control Manager	None	7036	N/A
INFO	4/21 12:00:00	EventLog	None	6013	N/A
INFO	4/21 11:49:04	Service Control Manager	None	7036	N/A
INFO	4/21 11:49:04	Service Control Manager	None	7035	gabriele
WARN	4/20 23:36:22	W32Time	None	50	N/A
WARN	4/20 22:54:53	Print	None	3	gabriele
INFO	4/20 22:54:51	Print	None	13	SYSTEM

Calculate Statistics

Calculate the distribution of the HTTP response status codes from your IIS log files:

C:\>LogParser "SELECT sc-status, COUNT(*) AS Times INTO Chart.gif FR OM <1> GROUP BY sc-status ORDER BY Times DESC" -chartType:PieExpl And and another states as desired:



System Requirements

Log Parser is compatible with the Windows® 2000, Windows® XP Professional, and Windows ServerTM 2003 operating systems.

© 2004 Microsoft Corporation. All rights reserved.

What's New in Log Parser 2.2

New Input and Output Formats:

XML Input Format

Reads XML files (requires the Microsoft® XML Parser (MSXML))

TSV Input Format

Reads tab- and space- separated values text files

ADS Input Format

Reads information from Active Directory objects

COM Input Format

Makes it possible to plugin user-implemented custom Input Formats

REG Input Format

Reads information from the Windows Registry

NETMON Input Format

Makes it possible to parse NetMon .cap capture files

ETW Input Format

Reads Event Tracing for Windows log files and live sessions

CHART Output Format

Creates chart image files (requires Microsoft Office 2000 or later)

TSV Output Format

Writes tab- and space- separated values text files

SYSLOG Output Format

Sends information to a SYSLOG server or to a SYSLOG-formatted text file

Improvements to the SQL Engine:

Exponential performance improvement in SELECT DISTINCT and GROUP BY queries

"WITH ROLLUP" functionality in the GROUP BY clause

"DISTINCT" in aggregate functions (when no group by clause is specified)

"PROPSUM(...) [ON <fields>]" and "PROPCOUNT(...) [ON <fields>]" aggregate functions

(these functions calculate the ratio between the SUM or COUNT functions on a field and the SUM or COUNT functions on the same field in a hierarchically higher group)

New functions:

- MOD
- BIT_AND, BIT_OR, BIT_NOT, BIT_XOR, BIT_SHL, BIT_SHR
- EXP10, LOG10
- ROUND, FLOOR
- QNTROUND_TO_DIGIT, QNTFLOOR_TO_DIGIT
- STRREPEAT
- IN_ROW_NUMBER, OUT_ROW_NUMBER
- ROT13
- EXTRACT_FILENAME, EXTRACT_EXTENSION, EXTRACT_PATH
- HEX_TO_ASC, HEX_TO_PRINT, HEX_TO_INT
- HEX_TO_HEX8, HEX_TO_HEX16, HEX_TO_HEX32
- IPV4_TO_INT, INT_TO_IPV4
- HASHSEQ, HASHMD5_FILE
- EXTRACT_PREFIX, EXTRACT_SUFFIX

• STRCNT

Introduced a "USING" clause for declaring temporary field-expressions

```
"BETWEEN" OPERATOR IN THE WHERE AND HAVING CLAUSES
```

```
"CASE" (simple-form) statement in the SELECT Clause
("SELECT CASE myField WHEN 'value1' THEN '0' WHEN 'value2' THEN
'1' ELSE '-1' END")
```

New date and time formats:

- 1 (milliseconds lower case 'L')
- n (nanoseconds)
- tt (AM/PM)
- ? (any character)

Fields and Aliases are now case-insensitive

Improvements to existing Input and Output Formats:

Added many new parameters to most of the Input and Output Formats

The NCSA input format now parses also *combined* and *extended* NCSA log files

Added "EventCategoryName" and "Data" fields to the EVT input format

The "-recurse" options of most input formats now specify a maximum subdirectory recursion level

The csv Input and Output Formats now support CSV files with doublequoted strings

Added "FileVersion", "ProductVersion", "CompanyName", etc. fields to the FS input format

Allowed '*' and '?' wildcards in the site name specifications for all the IIS input formats

("SELECT * FROM <mysite*.com>")

Allowed URL's as the input path of all text-based input formats ("SELECT * FROM http://www.adatum.com/table.csv")

Allowed use of environment variable names in the TPL output format sections, and added a SYSTEM_TIMESTAMP variable

Performance improvement in the EVT input format when reading from local and remote event logs

All the property names of the input and output format COM objects now match the command-line names

General improvements:

Added the possibility to specify parameters in .sql files

("logparser -file:myquery.sql?param1=value1+param2=value2")

Input I/O performance improvement for text files

Added the possibility to permanently override the default values of global options, input format options, and output format options

("logparser -e:10 -o:NAT -rtp:-1 -savedefaults")

© 2004 Microsoft Corporation. All rights reserved.

Conceptual Overview

This section provides information on the operational mechanisms of Log Parser.

- Log Parser Architecture: Describes the internal architecture of Log Parser.
- <u>Records</u>: Describes the data that Log Parser processes when working with Input and Output Formats.
- <u>Commands and Queries</u>: Describes how Log Parser commands are structured, and how you specify queries in a command.
- <u>Errors, Parse Errors, and Warnings</u>: Describes the run time errors that can be generated by Log Parser when executing a command.

© 2004 Microsoft Corporation. All rights reserved.

Log Parser Architecture

Log Parser is made up of three components:

- Input Formats are generic record providers; records are equivalent to rows in a SQL table, and Input Formats can be thought of as SQL tables containing the data you want to process.
 Log Parser's built-in Input Formats can retrieve data from the following sources:
 - IIS log files (W3C, IIS, NCSA, Centralized Binary Logs, HTTP Error logs, URLScan logs, ODBC logs)
 - Windows Event Log
 - Generic XML, CSV, TSV and W3C formatted text files (e.g. Exchange Tracking log files, Personal Firewall log files, Windows Media® Services log files, FTP log files, SMTP log files, etc.)
 - Windows Registry
 - Active Directory Objects
 - File and Directory information
 - NetMon .cap capture files
 - Extended/Combined NCSA log files
 - ETW traces
 - Custom plugins (through a public COM interface)
- A SQL-Like Engine Core processes the records generated by an Input Format, using a dialect of the SQL language that includes common SQL clauses (SELECT, WHERE, GROUP BY, HAVING, ORDER BY), aggregate functions (SUM, COUNT, AVG, MAX, MIN), and a rich set of functions (e.g. SUBSTR, CASE, COALESCE, REVERSEDNS, etc.); the resulting records are then sent to an Output Format.
- **Output Formats** are generic *consumers of records*; they can be thought of as SQL tables that receive the results of the data processing.

Log Parser's built-in Output Formats can:

• Write data to text files in different formats (CSV, TSV, XML, W3C,

user-defined, etc.)

- Send data to a SQL database
- Send data to a SYSLOG server
- Create charts and save them in either GIF or JPG image files
- Display data to the console or to the screen

Note: Transmitting data through a non-secure network might pose a serious security risk to the confidentiality of the information transmitted.

For more information on the security risks associated with nonsecure networks, see <u>Security Considerations</u>.



The Log Parser tool is available as a command-line executable (LogParser.exe) and as a set of scriptable COM objects (LogParser.dll). The two binaries are independent from each other; if you want to use only one, you do not need to install the other file on your computer.

© 2004 Microsoft Corporation. All rights reserved.

Records

Log Parser queries operate on **records** from an **Input Format**. Records are equivalent to *rows* in a SQL table, and Input Formats are equivalent to SQL tables containing the rows (data) you want to process.

Fields and Data Types

Each record generated by an Input Format is made up of a fixed number of **fields** (the *columns* in a SQL table), and each field is assigned a specific name and a specific **data type**; the data types supported by Log Parser are:

- Integer
- <u>Real</u>
- <u>String</u>
- <u>Timestamp</u>

Fields in a record can only contain **values** of the data type assigned to the field or, when the data for that field is not available, the <u>NULL</u> value.

For example, let's consider the <u>EVT Input Format</u>, which produces a record for each event in the Windows Event Log.

Using the command-line executable, we can discover the structure of the records provided by this Input Format by typing the following help command:

C:\>LogParser -h -i:ETW

The output of this command gives a detailed overview of the EVT Input Format, including a "Fields" section describing the structure of the records produced:

Fields:

EventLog (S) RecordNumber (I) TimeGenerated (T) From the rate of the bove, even inderstand that each spectral is made up of 15 fields and of each teaerables name (B) "Tissa Weittan" and always trapstations values of the tables TAMP data type. SID (S) Message (S) Data (S)

Record Structure

Some Input Formats have a fixed structure for their records (like the EVT Input Format used in the example above, or the <u>FS Input Format</u>), but others can have different structures depending on the values specified for their parameters or on the files being parsed.

For instance, the <u>NETMON Input Format</u>, which parses NetMon capture files, has a parameter ("fMode") that can be used to specify how the records should be structured. We can see the different structures when we add this parameter to the help command for the NETMON format. The first example shows the fields exported by the NETMON Input Format when its "field mode" is set to "TCPIP" (each record is a single TCP/IP packet), and the second example shows the fields exported by the NETMON Input Format when its "field mode" is set to "TCPConn" (each record is a full TCP connection):

C:\>LogParser -h -i:NETMON -fMode:TCPIP

Fields:

AS another example, the <u>CSV</u> hput Porniar, Which FrameBytes (files containing comma-separated values, creates its own structure by inspecting the input file for field names and types.⁽¹⁾ When using the help command with the CSV input Pormat, the "Fields" section shows no information on the record structure:

C:\>LogParser -h -i:NETMON -fMode:TCPConn

 Fields:

 Higher Sectors

 Content of the records produced when parsing that file:

 Structure of the records produced when parsing that file:

 SrcMAC (S)

 String Carter of the records produced when parsing that file:

 SrcPayloadBytes (I)
 SrcPayload (S)

 DstIP (S)
 DstPort (I)

 DstIP (S)
 DstPort (I)

 DstPayload (S)

 Fields:

Filename (SO) 200 Rom Monostoft (Corpolation Date (IT) ghts Message (S)

Commands and Queries

When using the command-line executable, Log Parser works on **commands** supplied by the user.

Each command has five distinct components:

- The Input Format to use;
- Optional parameters for the Input Format;
- The Output Format to use;
- Optional parameters for the Output Format;
- The SQL query that processes the records generated by the Input Format and produces records for the Output Format.

For example, let's consider the following simple command:

C:\>LogParser -i:EVT -fullText:OFF -o:CSV -tabs:OFF "SELECT * INTO out put.csv FROM SYSTEM"

The command above is structured as follows:

- The <u>EVT Input Format</u> is selected using the -i:<Input Format name> parameter;
- Its "fullText" parameter is set to the "OFF" value;
- The <u>CSV Output Format</u> is selected using the -o:<Output Format name> parameter;
- Its "tabs" parameter is set to the "OFF" value;
- The SQL query is "SELECT * INTO output.csv FROM SYSTEM", which specifies that all records generated from the System Event Log should be sent directly to the Output Format with no further processing.

In some cases, it might not be necessary to specify the Input Format. In the example command above, the value of the FROM clause is "SYSTEM", which is the name of a standard Windows Event Log; this name is automatically recognized by Log Parser as a candidate for the EVT Input Format, so we can avoid specifying the Input Format name altogether:

C:\>LogParser -fullText:OFF -o:CSV -tabs:OFF "SELECT * INTO output.csv FROM SYSTEM"

As examples of other values of FROM clauses that can be recognized by Log Parser, the <u>IISW3C Input Format</u> is selected automatically when the filename in the FROM clause starts with "ex" and has the ".log" extension, and the <u>XML Input Format</u> is selected automatically when the filename has the ".xml" extension.

The same applies to Output Formats: in the example command above, the filename in the INTO clause has the "csv" extension, thus selecting automatically the CSV Output Format; the same command can therefore be typed as:

C:\>LogParser -fullText:OFF -tabs:OFF "SELECT * INTO output.csv FROM SYSTEM"

When an Output Format is not specified, and the SQL query does not contain an INTO clause Log Parser automatically selects the <u>NAT Output</u> <u>Format</u>, which prints the results of the query to the console window.

These examples show the minimal Log Parser command is made up of the SQL query alone. In most cases the Input and Output formats can be deducted automatically from the INTO and FROM clauses of the query; however, it is a recommended good practice to always explicitly specify the Input and Output formats using the **-i** and **-o** parameters.

© 2004 Microsoft Corporation. All rights reserved.

Errors, Parse Errors, and Warnings

During the execution of a command, Log Parser can encounter three different types of run time errors: <u>Errors</u>, <u>Parse Errors</u>, and <u>Warnings</u>.

Errors

Errors are exceptional events occurring during the execution of a command that cause the command to abort.

Even though Errors can occur due to a large number of reasons, the most common causes can be categorized as follows:

- Invalid query syntax: the query specified in the command is invalid.
- **Input Format errors**: the specified Input Format has encountered an error that prevents it from generating input records. This could happen, for example, when the FROM clause specifies an entity (e.g. a file) that does not exist.
- **Output Format errors**: the specified Output Format has encountered an error that prevents it from consuming output records. This could happen, for example, when the INTO clause specifies an entity (e.g. a file) that cannot be written to.
- **Too many Parse Errors**: the specified Input Format has encountered too many <u>Parse Errors</u>, as specified by the "-e" command-line global parameter.
- Catastrophic errors: for example, Log Parser ran out of memory.

When an error occurs, the Log Parser command-line executable aborts the query execution and returns the error message and the error code. When an error occurs while using the Log Parser scriptable COM components, a COM exception is thrown containing the error message and the error code.

In most cases, the error code returned is the internal system error code that caused the error.

Parse Errors

Parse Errors are errors that occur while the selected **Input Format** generates the data on which the query operates.

Most of the times, as the name suggests, these errors are generated when a log has malformed entries (for example, when using the <u>IISW3C</u> <u>Input Format</u>), or when a system error prevents an Input Format from processing a specific entry in the data (for example, an "access denied" error on a file when using the <u>FS Input Format</u>).

In any event, the presence of a Parse Error indicates that the Input Format had to *skip* the data entry that caused the error; for example, when a Parse Error is encountered by the IISW3C Input Format while parsing a malformed line in the log, that line will be skipped and it will not be processed by the SQL engine.

Parse Errors do not generally cause early termination of the currently executing command, but rather, they are collected internally by the SQL engine and reported when the command execution is complete. This behavior can be controlled with the **-e** command-line global parameter. The value used with this parameter specifies a maximum number of Parse Errors to collect internally before aborting the execution of the command.

For example, if we execute a query on an IISW3C log file specifying "e:10", Log Parser will collect up to 10 Parse Errors during the execution of the command. If the IISW3C Input Format encounters 10 or less Parse Errors, the command will complete succesfully, and the collected Parse Errors will be reported in detail at the end of the execution. On the other hand, if the input log file contains more than 10 malformed log lines, the 11th Parse Error will cause the command to abort and return an <u>Error</u>.

The default value for this command-line parameter is **-1**, which is a

special value causing the SQL engine to ignore *all* Parse Errors and report only the total number of Parse Errors encountered during the execution of a command.

As an example, consider the following command, which parses an

IISW3C log file and writes all the input records to a CSV file:

C:\>LogParser -i:IISW3C -o:CSV "SELECT * INTO Output.csv FROM ex02 0528.log"

Let's assume that the "ex020528.log" log file contains 3 malformed log lines.

After executing the command above, the output will be as follows:

Task completed with parse errors.

Parse errors:

This But put to be a contend of the second successfully, but 3 Parse Errors have been encountered while processing the input data. Since the defaultivalue for the "-e" command-line parameter is -1, the SQL engine has ignored all these Parse Errors, keeping just their total count.

Elements processed: 997

If the "-e" operation the seggerse Errors to be reported in detail, we could specify a Evaluation the "-e" operation of the seggerse Errors to be reported in detail, we could specify a Evaluation the segmentation of the segmen

C:\>LogParser -i:IISW3C -o:CSV "SELECT * INTO Output.csv FROM ex02 0528.log" -e:10

In this case, the output would be:

Task completed with parse errors.

Parse errors:

The command still executed ausoesfully and this time the 28 marses Farors have been (spligging draine geported at the end of the execution.

LogFile "C:\Logs\ex020528.log", Row number 23, Value "2000" If warhadispesified 112 for the charaction attacted a Splanding way of the aborted the Splanding way of the constraint of of t

Log row terminates unexpectedly

The signification of the second secon

Too many parse errors - aborting

Barristècsors:

Error-while parsing field sc-status: Error parsing StatusCode "2b00": Extra Edemants (fs) dessed : 997 teger

EleogFints but pluogs 02020528.log", Row number 23, Value "2b00"

Execution dimension of OLion Seexands characters detected at the end of log entry

LogFile "C:\Logs\ex020528.log", Row number 118 Log row terminates unexpectedly LogFile "C:\Logs\ex020528.log", Row number 188

Statistics:

Elements processed: 182 Elements output: 181 Execution time: 0.01 seconds

Warnings

Warnings are exceptional events occurring during the execution of a command that require attention from the user.

There are only a few situations that could cause a warning, and these are handled differently depending on whether or not the warning arises during the execution of a command, or when the execution has completed.

When a warning is generated during the execution of a command, the command-line executable shows an interactive prompt to the user asking whether or not the execution should continue.

As an example, consider a command that writes output records to a CSV file.

The <u>CSV Output Format</u> "fileMode" parameter can be used to specify what action should be taken in case the output file already exists. The value "2" specifies that already existing output files should not be overwritten; when using this option, the CSV Output Format will raise a Warning when an already existing output file will not be overwritten:

C:\>LogParser -i:EVT -o:CSV "SELECT TOP 5 Message INTO Output.csv F ROM System" -fileMode:2

WARNING: File C:\LogSamples\Output.csv exists and it will not be overwritt en.

Where this promoting preases the user of the command allowing additional warnings to trigger the prompt again, aborting the execution of the command (in which case the command terminates with an $\underline{\text{Error}}$), or continuing the execution of the command ignoring additional warnings.

The interactive prompt can be controlled with the global **-iw** commandline parameter. This ON/OFF parameter specifies whether or not warnings should be ignored; the default value is "OFF", meaning that run time warnings will not be ignored and will trigger the interactive prompt. Specifying "ON", on the other hand, disables the interactive prompt, and run time warnings will be ignored and their total count will be reported when the command execution has completed:

C:\>LogParser -i:EVT -o:CSV "SELECT TOP 5 Message INTO Output.csv F ROM System" -fileMode:2 -iw:ON

Task completed with warnings.

Warnings:

1 warning occurred during processing and-line executable in a noninteractive script (e.g. in a script that has been scheduled to run automatically at specific times), you should always use "ON" for the Statistics' parameter, otherwise in the event of a run time warning the Log Parser command will stall waiting for a user to press a key in the

interactive prompt. Elements processed: 5

Elements output: 5

Execution time: 0.03 seconds Warnings that are generated when a command has completed are simply reported to the user.

For example, the "ignoreDspchErrs" parameter of the <u>SYSLOG Output</u> <u>Format</u> can be used to specify whether or not errors occurring while dispatching output records should be ignored and reported as warnings at the end of the execution.

The following example command uses the SYSLOG Output Format to send output records to a non-existing user:

```
C:\>LogParser -i:EVT -o:SYSLOG "SELECT TOP 5 Message INTO NonExis
```

tingUser FROM System" -ignoreDspchErrs:ON Since the specified user does not exist, the SYSLOG Output Format will encounter an error for each output record it will try to send to the user; the "ON" value for the "ignoreDspchErrs" tells the output format to ignore these errors and report all of them when the execution has completed:

Task completed with warnings.

Warnings:

The following dispatch errors occurred: <u>© 2004 Microsoft Corporation. All rights reserved</u>. The message alias could not be found on the network. (5 times) Statistics:

Elements processed: 5Elements output: 5Execution time: 0.02 seconds

Writing a Query

With Log Parser you use **Queries** written in a dialect of the SQL language to specify the operations that transform input <u>records</u> generated by an <u>Input Format</u> into output records that are delivered to an <u>Output Format</u>.

In this section we will cover the eight basic building blocks of the SQL-Like queries that you can use with Log Parser to perform different processing tasks.

© 2004 Microsoft Corporation. All rights reserved.

Basics of a Query

The most simple query that can be written with Log Parser specifies that all the <u>Input Records</u> generated by an <u>Input Format</u> are to be delivered to an <u>Output Format</u> with no intervening processing.

For example, let's assume that we want to visualize all the fields of all the events in the System Event Log. To perform this task, we first have to specify the <u>EVT Input Format</u> as the source of our input records, and we do so by using the "**-i:EVT**" command-line parameter.

Then, we can choose the <u>NAT Output Format</u> as the consumer of our output records, since this Output Format is specifically designed to print output records to the console window; we do so by using the "**-o:NAT**" command-line parameter. Finally, we specify the SQL query that performs the desired task; the complete command is as follows:

C:\>LogParser -i:EVT -o:NAT "SELECT * FROM System"

The query above contains the two basic building blocks of each possible query: the **<u>SELECT</u>** clause, and the **<u>FROM</u>** clause.

The SELECT clause is used to specify which input record fields we want to appear in the output records; in this example, the special "*" wildcard means "all the fields".

The FROM clause is used to specify which specific data source we want the Input Format to process. Different Input Formats interpret the value of the FROM clause in different ways; for instance, the EVT Input Format requires the value of the FROM clause to be the name of a Windows Event Log, which in our example is the "System" Event Log.

To be precise, the **INTO** clause should appear in every query as well. The INTO clause is used to specify the target we want the Output Format to write data to. In our example, we want the NAT Output Format to display results to the console window. This is accomplished by specifying "STDOUT" for the value of the INTO clause, as in the following example:

C:\>LogParser -i:EVT -o:NAT "SELECT * INTO STDOUT FROM System"

When a query does not specify an INTO clause, the NAT Output Format automatically selects "STDOUT" as its target, so in our example we can eliminate the INTO clause altogether.

Tip: When you use the NAT Output Format to display results to the console window, Log Parser prints 10 lines before pausing the printout and prompting the user to press a key to display the next 10 lines.

To override this behavior, you can use the "**-rtp**" parameter of the NAT Output Format to specify the number of lines to be printed before pausing; if you want to disable the pause altogether and have Log Parser display all the records in a single printout, use the "-1" value.

Selecting Specific Fields

When you execute the basic query above, Log Parser prints all the fields of all the events in the System Event Log to the console window. Most of the times, a printout of all of the 14 fields of the Event Log records might not be desired. For example, we might only want to see the time at which each event was generated, the type of the event, and the name of the source of the event.

To accomplish this, we have to substitute the "*" wildcard in the SELECT clause with a comma-separated list of the names of the fields we wish to be displayed. We can see the names of the fields in the EVT Input Format records by typing the following help command:

C:\>LogParser -h -i:EVT

The output of this command gives a detailed overview of the EVT Input Format, including a "Fields" section describing the structure of the records produced:

Fields:

EventLog (S) RecordNumber (I) TimeGenerated (T) From the fields tisting, we window that the fields way are interested in are vance of interested in the fields of the fi

C:\>LogParser -i:EVT -o:NAT "SELECT TimeGenerated, EventTypeName, S

ourceName FROM System"

Tip: Field names are case-insensitive.

Tip: If a field name contains spaces, you need to enclose it in square brackets ('[' and ']') for Log Parser to be able to recognize it.

The output of this command contains three columns, one for each of the fields we have selected:

TimeGenerated EventTypeNa	ame SourceName
---------------------------	----------------

This-example-illustrates-the-most-simple-transformation that you can a2004v@3w1th1t&64:55 WarsingSQLntangVa2jEintransforming an input record n2004-0β-off a 4n02n2e1rofbfield is into can Distput record made up of a subset o2004s@fieldst:02SQLnformstichis transformation is called projection.

2004-03-14 12:00:00 Information event EventLog

2004-03-14 00:41:47 Warning event W32Time

2004-03-13 22:17:00 Information event Service Control Manager

2004-03-13 22:06:48 Information event Service Control Manager

2004-03-13 22:06:48 Information event Service Control Manager

2004-03-13 12:00:00 Information event EventLog

2004-03-12 22:30:47 Information event Service Control Manager

Using Functions

Functions are very powerful elements of the Log Parser SQL-Like language that take values as arguments, do some processing, and return a new value.

The Log Parser SQL-Like language supports a wide variety of <u>functions</u>, including arithmetical functions (e.g. ADD, SUB, MUL, DIV, MOD, QUANTIZE, etc.), string manipulation functions (e.g. SUBSTR, STRCAT, STRLEN, EXTRACT_TOKEN, etc.), and timestamp manipulation functions (e.g. TO_DATE, TO_TIME, TO_UTCTIME, etc.).

Considering the previous example, assume that for the "TimeGenerated" field we only need to retrieve the date when an event has been generated, ignoring all of the time elements.

To do this, we need to modify the "TimeGenerated" field with the <u>TO_DATE</u> function, which takes a value of type <u>TIMESTAMP</u> and returns a new value of type TIMESTAMP containing only the year, day, and month elements:

C:\>LogParser -i:EVT -o:NAT "SELECT TO_DATE(TimeGenerated), EventT ypeName, SourceName FROM System" The output of this command is:

TO_DATE(TimeGenerated) EventTypeName SourceName

------ ------

by the <u>JOL JPPER CASE</u> at the two which the control Manager into a string with all popercase characters i event Service Control Manager

2004-03-13 Information event EventLog

2004L03Platser -i:EVilfoonNation'Stant Struce_Date (IIM and generated), TO_UP

PERGASE(FINTEGENERATE) TO KEN (FYPERCASE(EXTRA'C) SPOKEN(EVENT) MENANE,"0", ' ')) SourceName

2004-03-14	WARNING
2004-03-14	INFORMATION
2004-03-14	INFORMATION
2004-03-14	INFORMATION
2004-03-14	WARNING
2004-03-13	INFORMATION
Manager	
2004-03-13	INFORMATION
Manager	
2004-03-13	INFORMATION
Manager	
2004-03-13	INFORMATION
2004-03-12	INFORMATION
Manager	

W32Time Disk Disk EventLog W32Time Service Control Service Control

Service Control

EventLog Service Control

Specifying Constants

So far we have written SELECT clauses that specify both fields and functions.

There is a third kind of item that we could use in our gueries: **constants**. Constants are special elements in the Log Parser language that represent fixed values; just like the field values, constant values can be one of the Log Parser types: INTEGER, REAL, STRING, TIMESTAMP, and NULL. Constants can be specified in gueries in different ways, depending on their type.

Constant values of the INTEGER type are specified by simply typing their value; the following guery:

SELECT 242, SourceName FROM SYSTEM

would produce the following output:

242 W32Time

Constant values of the REAL type are specified exactly like the INTEGER values, but they are recognized as being of the REAL type by the presence of a decimal point: 242 W32Time

SELECT 242.7, SourceName FROM SYSTEM

242.700000 SourceName

242.700000 W32Time

STRING constants must be enclosed within single-quote characters: 242.700000 Disk

242.700000 EventLog SELECT 'MyConstant', SourceName FROM SYSTEM
242 700000 W32Time MyConstant SourceName

Spreycial share acter stime TRING constants can be specified by using characterstand quanties preceded by the '\' character.

For vexample, appingle-quote character can be specified as V, while a backslash character can be specified by N:

MyConstant W32Time

SELECT 'Contains \' a quote', 'Contains \\ a backslash', SourceName FROM S YSTEM

'Contains 'a quote' 'Contains \ a backslash' SourceName

Incadditionaltripedsoppossible to aperity any 321160DE character using the hexadecimal representation of thexadecimal representation of thexad

For the second of the second o

SELECT 'Contains \u0009 a tab', SourceName FROM SYSTEM

A NULL constant can be specified with the "NULL" keyword:

SELECT NULL, SourceName FROM SYSTEM

TIMESTAMP constants are specified in the following way:

TIMESTAMP('timestamp value', 'timestamp format')

For more information regarding timestamp values, constants, and format specifications, refer to the <u>Timestamp Reference</u>.

In the Log Parser SQL language, the three terms that can be specified in a SQL query (fields, functions, and constants) are collectively referred to

as field-expressions.

Aliasing Field-Expressions

Consider again one of the examples seen in this section:

C:\>LogParser -i:EVT -o:NAT "SELECT TO_DATE(TimeGenerated), TO_UP PERCASE(EXTRACT TOKEN(EventTypeName, 0, ' ')), SourceName FRO peName, 0, ' ')) SourceName We can see that for each field in the output record, the NAT Output Format prints a column header with the name of that field. Byodefealt, putput record fields are named with the full field expression textothabgenerates the or River and Reversion and Reversio record (field is "TO particity the mirrors and fight expression textrusor in Attra SELECT clause. EventLog 2004-03-14 WARNING W32Time We want and the man and the man and the second of the seco using an Alias. In ondersto alias a field expression in the SELECT clause, we can use the AS keyword followed by the new name: 2004-03-13 **INFORMATION** Service Control ManageParser -i:EVT -o:NAT "SELECT TO_DATE(TimeGenerated) AS Date 2020/4:43Bed 3TO UPPINEOR SEATER STRACT TOKEN (Event Ty Event and ego, '')) Service Control AS A CONTRACT SOLUTION AND A CONTRACT OF A C Manager ____ 2004-03-14 WARNING W32Time Afiasing a field-expression means assigning a name to it; as we will see later, this name can also be used anywhere else in the query as a shortcut that refers to the original field expression. 2004-03-14 WARNING W32Time 2004-03-13 INFORMATION Service Control Manager 2004-03-13 INFORMATION Service Control Manager INFORMATION Service Control Manager 2004-03-13 2004-03-13 INFORMATION Service Control Manager 2004-03-13 INFORMATION EventLog 2004-03-12 INFORMATION Service Control Manager

Filtering Input Records

When retrieving data from an Input Format, it is often needed to filter out unneeded records and only keep those that match specific criteria.

For example, consider the simple command seen in the previous section, which returns selected fields from all of the events in the System event log:

C:\>LogParser -i:EVT -o:NAT "SELECT TimeGenerated, EventTypeName, S ourceName FROM System"

TimeGenerated EventTypeName SourceName

Let's 4003 1215 584 576 55 at the age only interested in the events generated by the discrete the second se

Tomacomplish: this task own and the basic building block of the Loop Parsau SQ. bd: the Handwage the WHERE clause.

2004-03-14 00:41:47 Warning event W32Time T2004-03-14 00:41:47 Warning event W32Time T2004-03-13-22:17:50 is used to specify a boolean expression that must be satisfied by an input record for that record to be output alloger satisfied by an input record for that record to be output alloger alloger in SQL05-13-22:10:48 record for that record to be output alloger 12:004-03-14 00:41:47 Warning event Service Control Manager ranse is a transferration with the went Service Control Manager

Using the WHERE clause, we can rewrite the previous command as follows:

C:\>LogParser -i:EVT -o:NAT "SELECT TimeGenerated, EventTypeName, S

ourceName FROM System WHERE SourceName = 'Service Control Manager' " **Tip**: The WHERE clause must immediately follow the FROM clause.

The output of this command is:

TimeGenerated EventTypeName SourceName

Leo's 4 and 1 yze 2011 detail the Miller & classe we echinthis manageple. The boolean condition that we have used is a very simple one: we only w2004463st3i2put6:48drdormation espective this Contitle Massager exact value o205403ct3C2r0fo48/lanagertioTcespective this Contitle Mawagerave used the "=200468off220peratom for the netwoor Stand being the Manager field 4a03 t22 cight 32pterform the ing version for the Manager 2004-03-12 21:09:14 Information event Service Control Manager

Complex Conditions

Conditions specified in the WHERE clause can be more complex, making use of comparison operators (such as ">", "<=", "<>", "LIKE", "<u>BETWEEN</u>", etc.) and boolean operators (such as "AND", "OR", "NOT").

For example, we might only want to see two kinds of events:

- Events generated by the "Service Control Manager" source whose EventID is greater than or equal 7024;
- Events generated by the "W32Time" source.

To accomplish this, the query can be written as follows:

SELECT TimeGenerated, EventTypeName, SourceName FROM System WHERE (SourceName = 'Service Control Manager' AND EventID >= 7024) OR

As another example, we might want to see all the events that have been logged in the past 24 hours.

Translated into WHERE terms, this means that we only want to see records whose "TimeWritten" field is greater than or equal the current local time minus 1 day:

SELECT * FROM System

```
WHERE TimeWritten >= SUB( TO_LOCALTIME(SYSTEM_TIMESTAMP()), TIME TAKEN (), TIME T
```

day 2 of month 1 of year zero, i.e. 24 hours after the origin of time.

To see security events whose "Message" field contains the word "logon", we can use the <u>LIKE</u> operator, which tests a STRING value for case-insensitive pattern matching:

WHERE Message LIKE '%logon%'

If we want to retrieve events with an ID belonging to a specific set of values, we can use the **IN** operator followed by a list of the desired "EventID" values:

SELECT * FROM Security WHERE EventID IN (547; 541; 540; 528) Tip: With the IN operator, single values are separated by the semicolon character.

On the other hand, if we want to retrieve events with an ID belonging to a specific *range* of values, we can use the **<u>BETWEEN</u>** operator as follows:

SELECT * FROM Security WHERE EventID BETWEEN 528 AND 547

© 2004 Microsoft Corporation. All rights reserved.

Sorting Output Records

A commonly used building block of SQL queries is the **ORDER BY** clause.

The ORDER BY clause can be used to specify that the output records should be sorted according to the values of selected fields.

In the following example, we are using the <u>FS</u> Input Format to retrieve a listing of the files in a specific directory, sorting the listing by the file size:



C:\>LogParser -i:FS -o:NAT "SELECT Name, Size, CreationTime FROM C:\

MyDirectory*.* ORDER BY Size CreationTime"

0 2004-05-24 08:14:07.221

Since the sort operation is performed on output records, the Log Parser SOL-Like language requires that field expressions appearing in the ORDER BY clause must also appear in the SELECT clause. In Sther Words, the set of fleid expressions in the ORDER BY clause must be a subset of the field expressions in the SELECT clause. Thus, the following example is NOT correct: 812 caspol.exe.config 353 2004-05-24 08:14:20.920 ConfigWizards.exe.config 353, 2004-05-24 08:14:21.21 SELECT SourceName, EventID FROM System cytres.exe.config 353, 2004-05-24 08:14:21.251 ORDER BY TimeGenerated

On the other hand, the following example IS correct:

SELECT SourceName, EventID, TimeGenerated FROM System **ORDER BY TimeGenerated**

© 2004 Microsoft Corporation. All rights reserved.

Aggregating Data Within Groups

All the query examples that we have seen so far share a common characteristic: the values of each output record were built upon the values of a single input record.

Sometimes, however, we might need to *aggregate* multiple input records together and perform some operation on *groups* of input records.

To accomplish this task, the Log Parser SQL-Like language has a special set of functions that can be used to perform basic calculations on multiple records. These <u>aggregate functions</u> (also referred to as "SQL functions") include **SUM**, **COUNT**, **MAX**, **MIN**, and **AVG**.

Aggregating Data

To show a classic example of the use of aggregate functions, assume that given an IIS W3C log file, we want to calculate the total number of bytes sent by the IIS server during the whole period recorded in the log file.

Considering that the number of bytes sent by the IIS server for each HTTP request is logged in the "sc-bytes" field, our command will look like the following example:

C:\>LogParser -i:IISW3C -o:NAT "SELECT SUM(sc-bytes) FROM ex040528 .log"

Since the SELECT clause of this query makes use of the **SUM** aggregate function, the query will automatically aggregate all the input records, and calculate the sum of all the values of the "sc-bytes" field across all the input records; the output of this command will then look like the following output:

SUM(sc-bytes)

As the result of the query is a single output record, containing a single value calculated across all the input records.

As another example, we might want to calculate how many requests have been logged in the log file.

Considering that each log file entry represents a single HTTP request, this task can be accomplished by simply counting how many input records are logged in the file:

C:\>LogParser -i:IISW3C -o:NAT "SELECT COUNT(*) FROM ex040528.log

The example above makes use of the COUNT aggregate function. When used with the special "*" argument, the COUNT function returns the total

number of input records processed by the query.

If we want to calculate how many requests satisfy a particular condition, for example how many requests were for an ASP page, we can add a <u>WHERE</u> clause to the query, and the COUNT function will only count input records satisfying the WHERE condition:

SELECT COUNT(*) FROM ex040528.log WHERE EXTRACT_EXTENSIO N(cs-uri-stem) LIKE 'asp'

Creating Groups

In the examples above, we have been using aggregate functions to calculate a value across *all* the input records; sometimes, however, we might want to calculate values across *groups* of input records.

As an example, we might want to calculate the total number of bytes sent by the IIS server for each URL. To perform this task, we need to divide all the input records into groups according to the URL requested, and then use the SUM aggregate function separately on each group.

This can be accomplished by using another building block of the Log Parser SQL language: the **<u>GROUP BY</u>** clause.

The GROUP BY clause is used to specify which fields we want the group subdivision to be based on; after the input records have been divided into these groups, all the aggregate functions in the SELECT clause will be calculated separately on each of these groups, and the query will return an output record for each group created.

Using the GROUP BY clause, our example query and its output will look like this:

SELECT cs-uri-stem, COUNT(*) FROM ex040528.log GROUP BY cs-uri-ste m

cs-uri-stem COUNT(*)

/Home/default.asp 5

To make another example, assume that we want to calculate how many reduests have been served for each page type (ASP, html, CSS, etc.). First of all, we need to create separate groups according to the extension of the URL; after this group subdivision has been done, we can calculate a COUNT(*) on each group:

SELECT EXTRACT_EXTENSION(cs-uri-stem) AS PageType, COUNT(*) FROM ex040528.log The output will look like:

FBCHBECOUNT(REL *)

If we sort the output above according to the number of requests for each group, we will be creating a list showing the most requested page types fi<u>£</u>§‡: 585

25 exe

6SfLECT**4E**XTRACT_EXTENSION(cs-uri-stem) AS PageType, COUNT(*) A **Swffage** TylpeHits

The OMPEND Will 200 k dike:

MRDUPIBY PageType

BERBERAPAPATET

asp---- 5------

gif **58**5

Gloups can also be built on multiple fields, thus creating a hierarchy of groups. ¹¹⁵ ipg 77

jpg

For example, consider the following query:

CSS

SELECT EXTRACT_EXTENSION(cs-uri-stem) AS PageType, sc-status, CO ŬNT(*1

The ondersome sources according to the requested page type, and with the second states are created according to the HTTP status sent by the IIS server for the group page type; the aggregate function "COUNT" will then be calculated on each sub-group. The output will look like:

PageType sc-status PageTypeHits

htm 304 79

It simplifiant to note a particular language constraint derived from the use of the GROUP BY clause.

Whenever a query contains a GROUP BY clause, its <u>SELECT</u> clause can only contain any of the following: swf 200

• gAggragate functions

• csield 40 Apres gions appearing also in the GROUP BY clause, or deriving

htmom taeofield a pressions used in the GROUP BY clause

• C Sonstants 3

jpg 200 17

In the powords 1 the following example is a correct query:

jpg 304 60

SvEfLEC**3**0/hello/8TO_UPPERCASE(cs-uri-stem), COUNT(*), SUM(sc-bytes) **F**SFOM4030405**2**8.log

In fact the SELECT chause in the example above contains:

- •dA co59tant (1hello");
- am field expression ("TO_UPPERCASE(cs-uri-stem)") whose argument ^{js}appears in the GROUP BY clause;
- ^{class} 304 Two aggregate functions.

js 200 4

htm 404 2

Housever the following example is NOT a correct query:

nsf 304 9

SELECP date, COUNT(*), SUM(sc-bytes)

FROM ex040528.log

The SEPERT colarise in the example above contains a field-expression ("date") that does not appear in the GROUP BY clause.

The following example is also NOT a correct query:

SELECT TO_UPPERCASE(cs-uri-stem), COUNT(*), SUM(sc-bytes)

FROM ex040528.log

The SELECT clause.

The previous example can be corrected as follows:

SELECT SUBSTR(TO_UPPERCASE(cs-uri-stem), 0, 5), COUNT(*), SUM(s c-bytes)

FROM ex040528.log GROUP BY SUBSTR(TO_UPPERCASE(cs-uri-stem), 0, 5)

Calculating Percentages

When working with groups and aggregate functions, it is often needed to represent an aggregate value as a percentage, rather than as an absolute value.

We might want, for example, to calculate the number of hits per page type from a Web server log as a percentage relative to the *total* number of hits, rather than as the absolute number itself.

Consider the previous example query, that calculates the count of hits per requested page type:

SELECT EXTRACT_EXTENSION(cs-uri-stem) AS PageType, COUNT(*) FROM ex040528.log

PAROLIA *)

If we wanted to calculate the percentage of hits for each group, we would need to divide the number of hits within each group by the total number of fits in the single whole log file; however, the use of the GROUP BY clause restricts gach aggregate function to operate within the single groups, thus making it impossible to calculate at the same time the total number of hits aggregss all groups.

jpg 77 html 1

Tollworkaround this problem, we use two special aggregate functions a stallable in the Log Parser SQL language: **PROPCOUNT** and **PROPSUM**.

When used in their basic forms, these functions calculate the ratio of the COUNT or ADD aggregate functions within a group to the COUNT or ADD aggregate functions on all of the input records.

Using the PROPCOUNT function, we can change the query above as follows:

SELECT EXTRACT_EXTENSION(cs-uri-stem) AS PageType, PROPCOUN T(*) And obtain:

FROM ex040528 OUNT(ALL *) GROUP BY PageType

Tete how or easy of the can multiply the aggregate function values by \$100:0.022000

gif 0.585000

STALE COT CE2XCOR ACT_EXTENSION(cs-uri-stem) AS PageType, MUL(PROP 6: 0000.0) AS PageTypeHits

FREE WEEPERS

FBOUD.BZZRAGeType

Firm the 1.950000 of this query we can infer that, for example, requests to "css" people (requests in this log

5895050000 fibesp 020300000 **jé**xe filass 1402050000 swf 1.100000 7.700000 jpg 0.100000 html 0.100000 dll 0.500000 asp 1.100000 js 0.500000 class

Calculating Percentages Across Multiple Group Hierarchies

The examples above show the basic form of the PROPCOUNT and PROPSUM functions, which calculates the percentage of an aggregate function within a group relative to *all* of the input records. However, it is also possible to use the PROPCOUNT and PROPSUM functions to calculate percentages relative to hierarchically higher groups. To do so, we can use the **ON** keyword after the PROPCOUNT or PROPSUM function name followed by a list of the GROUP BY fieldexpressions identifying which hierarchically higher group we want the percentage to be relative to.

Consider one of the previous examples, in which we calculated the total number of hits per page type per HTTP status code, modified to show percentages rather than absolute numbers:

SELECT EXTRACT_EXTENSION(cs-uri-stem) AS PageType, sc-status, MU L(PROPCOUNT(*), 100.0) AS Hits

FROM BEQ4052811 PEits

GROUP.BY_PageType, sc-status

The DHills of the Regeneration of hits for a page type and HTTP status code relative to the total number of hits.

class 304 0.400000

If we wanted to calculate the percentage of hits for a page type and HTTP status code relative to the number of hits for that page type (i.e. the distribution of HTTP status codes within each page type), we would have written the quantum as follows:

exe 200 2.500000

SHELEGTOEXTRASOFORTENSION(cs-uri-stem) AS PageType, sc-status, MU LAPROPAOUNE(MONO(PageType), 100.0) AS Hits

TER OMPRIS DAOSPIS DE COMPANDA DE COMPANDA

ROUPBY Page Type Ogc-status

REDEXE BERGE BERGE

Vision and the second status code of 304.

jakass 2004 B7.0000000

Hase wadrave source of the second of the sec

ezzares 2000 . 219025641

As another example, we can modify the previous example query to create groups based on the time the request was made at (quantized at 20-second intervals), the page type, and the HTTP status code: htm 304 68.695652

htm 404 1.739130 SELECT QUANTIZE(time, 20) AS Interval, EXTRACT_EXTENSION(cs-uri html 404 100.000000 -stem) AS PageType, sc-status

For the percentage of hits relative to the total number of hits: percentage of hits relative to the total number of hits: percentage of hits relative to the total number of hits: percentage of hits relative to the total number of hits: percentage of hits relative to the total number of hits:

nsf 302 0.704225 SELECT QUANTIZE(time, 20) AS Interval, EXTRACT_EXTENSION(cs-uri nsf 304 6.338028 -stem) AS PageType, sc-status, nst 403 2.112676

FOO! 1994 BED P 2000 LISE TACE CARDING FOR Statistics an ple othet during the "OR 29, 2000 CB and a statistic control of the statistic control of

In the Barbard Interimite Rade Transforments that 1 pages one turning the HTTP status and these requests requests requests requests in the log.

00:28:40 gif 404 20.000000 2.941176 0.200000 The example above shows that a PROPEOUNT or BROPSUM function with 20:40 N keyword is logically equivalent to using the ON keyword followed by an empty list of GROUP 4875 field examples sions, meaning that the percentage to be calculated should be relative to the highest hierarchical group identified by the field examples sion inclusion input records.

00:29:00 css 404 57.142857 0.867679 0.400000

922209 QUANTORE (tille, 20) AS 9226, SC-4Q4, 20000 1.952278 0.900000

HOW APROLOGICATION PROPERTY IS CORRECT PROPERTY PROPERT

OR29:09 Byminter 200, Page 7,090,909 starts 3796 1.500000

9ይደደረዋ መuan ምምድር (time, 2003, 45 በዓ. 67375, ት. አ. ምጽዓረዋ EXTENSION (cs-uri ቢዮሩት) ባለታ ምድር ተቀበረ sc-state 7, 727 0.216920 0.100000

00: RACOO (jpg OP 204 UNT (5) 00 00 00 ter 819 357 stat 39, 000.0) AS Hits2,

PR:09.0028.00g	5Ò.Ó0000Ò	0.433839	0.200000	
$\theta \theta \partial \theta \partial \theta \theta \partial \theta \partial \theta \partial \theta \partial \theta \partial \theta \partial \theta \partial$		0 400000	0 200000	

GROUP BY In	ter≁al, I	>2600000 0.200000
00:29:00 nsf	200	94.339623 10.845987 5.000000
00:29:00 nsf	403	5.660377 0.650759 0.300000
00:29:00 swf	200	50.000000 0.433839 0.200000
00:29:00 swf	304	50.000000 0.433839 0.200000
00:29:20 NSF	200	100.000000 2.127660 0.300000
00:29:20 asp	200	$100.000000 \ 0.709220 \ 0.100000$
00:29:20 class	304	$100.000000 \ 0.709220 \ 0.100000$
00:29:20 css	304	60.000000 2.127660 0.300000
00:29:20 css	404	40.000000 1.418440 0.200000
00:29:20 exe	200	100.000000 2.836879 0.400000
00:29:20 gif	304	97.142857 48.226950 6.800000
00:29:20 gif	404	2.857143 1.418440 0.200000
00:29:20 htm	200	15.789474 2.127660 0.300000
00:29:20 htm	304	78.947368 10.638298 1.500000

00:29:20 htm	404	5.263158 0.709220 0.100000
00:29:20 jpg	200	$15.384615 \ 1.418440 \ 0.200000$
00:29:20 jpg	304	84.615385 7.801418 1.100000
00:29:20 js	200	50.000000 1.418440 0.200000
00:29:20 js	304	50.000000 1.418440 0.200000
00:29:20 nsf	200	$61.111111 \ 7.801418 \ 1.100000$
00:29:20 nsf	302	5.555556 0.709220 0.100000
00:29:20 nsf	304	33.333333 4.255319 0.600000
00:29:20 swf	304	100.000000 2.127660 0.300000

Filtering Groups

Consider again one of the previous examples, in which we used the COUNT aggregate function to calculate the number of times each page type has been requested:

SELECT EXTRACT_EXTENSION(cs-uri-stem) AS PageType, COUNT(*) A S PageTypeHits

FRQM/6894Q548-bestits

GROUP BY Page Type

Lefts Diago Base Strage Type Hive Diffs Only interested in seeing page types that have been requested 10 times or more.

htm 115

At first glance, it might seem that we could use a WHERE clause with a condition on the value of the COUNT aggregate function to filter out the undesired groups.

However, we have seen that the WHERE clause is used to filter input records, which means that this clause is evaluated *before* groups are created₅. For this reason, use of aggregate functions is not allowed in the WHERE clause.

dll 1

html 1

The task at hand can be accomplished by using the **HAVING** clause. The HAVING clause works just like the WHERE clause, with the only difference being that the HAVING clause is evaluated *after* groups have been created, which makes it possible for the HAVING clause to specify aggregate functions.

Tip: The HAVING clause must immediately follow the GROUP BY clause.

Using the HAVING clause, we can write the example above as:

SELECT EXTRACT_EXTENSION(cs-uri-stem) AS PageType, COUNT(*) A S PageTypeHits AFRICIDITE: 040528.log GROUP BY PageType

HAVING Bage TypeHits >= 10 Page Type Page TypeHits >= 10 ORDER BY Page TypeHits DESC

gif nsf	 585 142 © 2004 Microsoft Corporation. All rights reserved.
nsf	142
htm	115
jpg	77
exe	25
CSS	22
js	11
swf	11

Eliminating Duplicate Values

When working with information from logs, it is often desired to retrieve a list of some values where each element in the list appears only once, regardless of the number of times the same value appears in the original data.

As an example, consider the following query, which extracts all the domain accounts that have logged on a computer from the "Security" event log:

SELECT RESOLVE_SID(Sid) AS Account

FROM \\TESTMACHINE1\Security

The put of this pression 52 list of all the domain accounts appearing in each "Logon" event:

Account

If we are interested in catric wire a dist in which each account name appears to the sector of the s

NT AUTHORITY\NETWORK SERVICE

SESEDOMISINNES RESERVE_SID(Sid) AS Account

NROAND WHESHINNA COOM 1 Stern fye

And Hald Him Orally IDQ 640; SHERVICE

TESTDOMAIN\TESTUSER1

TESTIDOMAIN\TESTUSER2

NT-AUTHORITY\LOCAL-SERVICE--

The SANSTHMATINE AND A query should complete that the output of a query should complete the output records are discarded.

As another example, we might want to retrieve a list of all the browsers TESTDOMAIN TESTLISER2 used to request pages from our IIS server, with each browser appearing only once in the list:

SELECT DISTINCT cs(User-Agent) FROM <1> cs(User-Agent)

Itvisoalna/p.ossible patuse, the SPEST IN CWike wordvinside the COUNT aggragate of the patin patin pating of the count of different values appearing in the data.

Mozilla/4.0+(compatible:+MSIE+6.0;+Windows+NT+5.0;+T312461:+O3124 For example, the following query returns the total number of different browsers and the total number of different client IP addresses that Mozilla/4.0+(compatible:+MSIE+5.01;+Windows+NT+5.0) requested pages from our IIS server: Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.0)

Microsoft+Data+Access+Internet+Publishing+Provider+Cache+Manager SELECT COUNT(DISTINCT cs(User-Agent)) AS Browsers, Mozilla/2.0+(compatible;+MS+FrontPage+4.0) MSFrontPage/4.0 MSFrontPage/4.0 BROWSER Clients Microsoft+Data+Access+Internet+Publishing+Provider+DAV

356 **Tip** the Log Parser SQL-Like language, the DISTINCT keyword can be used inside aggregate functions only when the GROUP BY clause is not used.

© 2004 Microsoft Corporation. All rights reserved.

Retrieving a Fixed Number of Records

One of the most common log reports is a "TOP 10" list showing the top entries appearing in a ranking.

This is usually achieved with a query that calculates some aggregate function within groups, orders the groups by the value of the aggregate function, and then uses the **TOP** keyword in the <u>SELECT</u> clause to return only a few records at the top of the ordered output.

As an example, the following query returns the TOP 10 URL's requested from an IIS log file:

SELECT TOP 10 cs-uri-stem AS Url,

COUNT(*) AS Hits

EROM <1> Hits

GROUP BY Url

TOBLE A MINING TO BE A perfect candidate for the <u>CHART</u> Output Format; assuming that the following query is saved in the "querytop.sql" text file, the following comprand with generate an image file containing a chart of the one of the one is a containing a chart of

/images/address.gif 3609

SEASE CT_EQPS10 cs-uri-SEAOAS Url,

/npanews@CMMdNT(*) AS BB05

 Image: Second State
 Image: Second State

 Image: Second State
 Image: Second State</t



© 2004 Microsoft Corporation. All rights reserved.

Improving Query Readability

The functions available in the Log Parser SQL language make it possible to write complex queries operating on a very large number of possible transformations of the input fields; however, these complex queries might sometimes be cumbersome to write.

As an example, consider the task of writing a query that extracts from the Security event log all the users belonging to a specific domain that logged on this computer.

For the purpose of the example, let's also assume that we want the user names as lowercase strings, and that we are writing the query as a SQL file that takes a lowercase domain name as an input parameter. At first thought, the query would look like this:

SELECT EXTRACT_TOKEN(TO_LOWERCASE(RESOLVE_SID(Sid)), 1 , '\\') AS Username

To restrict the state of the st

EXTRACT_TOKEN(TO_LOWERCASE(RESOLVE_SID(Sid)), 0, '\\') €:%HogPainstenfid%hyquery.sql?domainname=tstdomain -i:EVT

When typing the query above, we had to repeat twice the whole expression that transforms the Sid input record field into a lowercase fully-qualified account name:

TO_LOWERCASE(RESOLVE_SID(Sid))

It would be easier if we could, in a certain sense, "assign" this expression to a "variable", and then use the variable when needed. We could definitely do that by aliasing the expression in the SELECT clause:

SELECT TO_LOWERCASE(RESOLVE_SID(Sid)) AS FQAccount,

EXTRACT_TOKEN(FQAccount, 1, '\\') AS Username Horever, the number of this query now contains an extraneous field - the fully-qualified account name: WHERE EventID IN (540: 528) AND

EXTRACT_TOKEN(FQAccount, 0, '\\') = '%domainname%'

Tetabyiate this problems the Log Parser SQL language supports the USING all testus 1 testus 1

The line standard SQL language element, is used to declare alige standard sqL language element, is used to declare alige standard sqL language element, is used to the difference that expressions in the USING clause will not appear in the output records (unless explicitly referenced in the SELECT clause).

With the USING clause, the query above can be written as follows:

SELECT EXTRACT_TOKEN(FQAccount, 1, '\\') AS Username USING TO_LOWERCASE(RESOLVE_SID(Sid)) AS FQAccount FROM Tip: The USING clause must immediately follow the SELECT clause. WHERE EventID IN (540; 528) AND The oBIFUR of ChisTople EVICOUS dook dike, the following is a mple output:

Username	
testusr1	© 2004 Microsoft Corporation. All rights reserved.
testusr1	
testusr2	
testusr3	

Advanced Features

Log Parser offers a unique set of features that enhance its flexibility in the most common log processing scenarios.

These features include:

- Parsing Input Incrementally: some input formats allow Log Parser to parse incrementally logs that grow over time.
- <u>Multiplexing Output Records</u>: some output formats allow the output records of a query to be written to different targets, depending on the values of selected output record fields.
- Converting File Formats: due to its architecture, Log Parser can be easily used to convert log files from a format to another.
- <u>Custom Plugins</u>: Log Parser allows users to develop their own custom input formats, and use them with either the Log Parser command-line executable, or with the Log Parser scriptable COM components.

© 2004 Microsoft Corporation. All rights reserved.

Parsing Input Incrementally

Log Parser is often used to parse logs that grow over time.

For example, the IIS logs and the Windows Event Log are continuously updated with new information, and in some cases, we would like to parse these logs periodically and only retrieve the new records that have been logged since the last time.

This is especially true for scenarios in which, for example, we use Log Parser to consolidate logs to a database in an almost real-time fashion, or when we use Log Parser to build a monitoring system that periodically scans logs for new entries of interest.

For these scenarios, Log Parser offers a feature that allows sequential executions of the same query to only process new data that has been logged since the last execution.

This feature can be enabled with the **iCheckPoint** parameter of the following input formats:

- <u>IISW3C</u>
- <u>NCSA</u>
- <u>IIS</u>
- <u>HTTPERR</u>
- URLSCAN
- <u>CSV</u>
- <u>TSV</u>
- <u>EVT</u>
- <u>TEXTLINE</u>
- <u>TEXTWORD</u>

The "iCheckPoint" parameter is used to specify the name of a "checkpoint" file that Log Parser uses to store and retrieve information about the "position" of the last entry parsed from each of the logs that appear in a command.

When we execute a command with a checkpoint file for the first time (i.e. when the specified checkpoint file does not exist), Log Parser executes the query normally and processes all the logs in the command, saving for

each the "position" of the last parsed entry to the checkpoint file. If later on we execute the same command specifying the same checkpoint file, Log Parser will parse again all the logs in the command, but each log will be parsed starting *after* the entry that was last parsed by the previous command, thus producing records for new entries only. When the new command execution is complete, the information in the checkpoint file is updated with the new "position" of the last entry in each log.

Note: Checkpoint files are updated only when a query executes succesfully. If an <u>error</u> causes the execution of a query to abort, the checkpoint file is not updated.

To make an example, let's assume that the "MyLogs" folder contains the following text files:

- Log1.txt, 50 lines
- Log2.txt, 100 lines
- Log3.txt, 20 lines
- Log4.txt, 30 lines

Let's also assume that we want to parse these text files incrementally using the <u>TEXTLINE Input Format</u>, which returns an input record for each line in the input text files.

In order to parse these logs incrementally, we specify the name of a checkpoint file, making sure that the file does not exist prior to the command execution. Our command would look like this:

logparser "SELECT * FROM MyLogs*.*" -i:TEXTLINE -iCheckPoint:myCh eckPoint.lpc

When this command is executed for the first time, Log Parser will return all the 200 lines from all of the four log files, and it will create the "myCheckPoint.lpc" checkpoint file containing the position of the last line in each of the four log files.

Tip: When the checkpoint file is specified without a path, Log Parser will create the checkpoint file in the folder currently set for the %TEMP% environment variable, usually "\Documents and Settings\ <user name>\Local Settings\Temp".; Let's now assume that the "Log3.txt" file is updated, and that ten new lines are added to the log file.

At this moment, the log files and the information stored in the checkpoint file will look like this:

Log FilesCheckpoint fileLog1.txt, 50 linesLog1.txt, line 50Log2.txt, 100 linesLog2.txt, line 100Log3.txt, 30 linesLog3.txt, line 20Log4.txt, 30 linesLog4.txt, line 30

If we execute again the same command, Log Parser will use the "myCheckPoint.lpc" file to determine where to start parsing each of the log files, and it will only parse and return the ten new lines in the "Log3.txt" file. When the command execution is complete, the "myCheckPoint.lpc" checkpoint file is updated to reflect the new position of the last line in the "Log3.txt" file.

If now a new "Log5.txt" file is created containing ten lines, the log files and the information stored in the checkpoint file will look like this:

Log FilesCheckpoint fileLog1.txt, 50 linesLog1.txt, line 50Log2.txt, 100 linesLog2.txt, line 100Log3.txt, 30 linesLog3.txt, line 30Log4.txt, 30 linesLog4.txt, line 30

Log5.txt, 10 lines not recorded

If we execute again the command, Log Parser will only parse the new "Log5.txt" file, returning its ten lines.

As another example showing how the checkpoint file is updated, let's assume now that the "Log2.txt" file is deleted.

The log files and the information stored in the checkpoint file will now look like this:

Log Files Checkpoint file Log1.txt, 50 lines Log1.txt, line 50

non-existing Log2.txt, line 100

Log3.txt, 30 lines Log3.txt, line 30 Log4.txt, 30 lines Log4.txt, line 30 Log5.txt, 10 lines Log5.txt, line 10

When we execute the command, Log Parser will detect that there are no new entries to parse, and it will return no records. However, upon updating the checkpoint file, it will determine that the "Log2.txt" file doesn't exist anymore, and it will remove all the information associated with the log file from the checkpoint file, which will now look like this:

Log Files Checkpoint file Log1.txt, 50 lines Log1.txt, line 50 Log3.txt, 30 lines Log3.txt, line 30 Log4.txt, 30 lines Log4.txt, line 30 Log5.txt, 10 lines Log5.txt, line 10

At this moment the checkpoint file does not contain anymore information on the "Log2.txt" file; should a new "Log2.txt" file appear again for any reason, a subsequent command would treat the file as a new file, and all of its entries would be parsed from the beginning of the file.

As a last example, let's now assume that the "Log1.txt" file is updated, but this time its size shrinks and it ends up containing ten lines only. The log files and the information stored in the checkpoint file will now look like this:

Log Files Checkpoint file

Log1.txt, 10 lines Log1.txt, line 50

Log3.txt, 30 lines Log3.txt, line 30

Log4.txt, 30 lines Log4.txt, line 30

Log5.txt, 10 lines Log5.txt, line 10

When we execute the command, Log Parser will detect that the size of the "Log1.txt" file has changed, but instead of growing larger, the file is actually smaller. In this situation, Log Parser assumes that the file has been *replaced* with a new one, and it will parse it as if it was a new file, returning all of its ten entries.

After the command execution is complete, the "myCheckPoint.lpc"

checkpoint file is updated to reflect the new situation, and the log files and the information stored in the checkpoint file will look like this:

Log Files Checkpoint file Log1.txt, 10 lines Log1.txt, line 10 Log3.txt, 30 lines Log3.txt, line 30 Log4.txt, 30 lines Log4.txt, line 30 Log5.txt, 10 lines Log5.txt, line 10

Incremental Parsing and Aggregated Data

It's important to note that the checkpoint file only records information about the files being parsed; it does not record information about the *query* being executed.

In other words, when we execute a query multiple times on a set of growing files using a checkpoint file, each time the query results are calculated on the new entries only. This means that queries using aggregated data need to be handled carefully when used with checkpoint files.

As an example, consider again the four text files in the first scenario above, and the following command:

logparser "SELECT COUNT(*) AS Total FROM MyLogs*.*" -i:TEXTLINE -iCheckPoint:myCheckPoint.lpc

When the command is executed for the first time, the "Total" field in the output record returned by the query will be equal to 200, that is, the total number of lines in the four log files.

As in the first example, let's now assume that the "Log3.txt" file is updated, and that ten new lines are added to the log file. When we execute the command again, the "Total" field in the output record returned by the query will be now equal to 10, the total number of *new* lines in the four log files, and not to 210, as one would expect from the total number of rows.

In cases where it is desirable to calculate aggregated data across multiple executions of the same query when using incremental parsing, a possible solution is to save the partial results of each query to temporary files, and then aggregate all the partial results with an additional step. Using the example above, we could save the result of the first query ("200") to the "FirstResults.csv" file, and the result of the second query ("10") to the "LastResults.csv" file. The two files could then be consolidated into a single file with a command like this:
logparser "SELECT SUM(Total) FROM FirstResults.csv, LastResults.csv" -i: CSV

© 2004 Microsoft Corporation. All rights reserved.

Multiplexing Output Records

Many Log Parser output formats allow the user to specify *multiple* files as the target to which output records are written to.

This is achieved by using '*' wildcard characters in the filename specified in the <u>INTO</u> clause; during the execution of the query, the first fields in each output record substitute the wildcard characters to determine the resulting filename to which the output records with the remaining fields are written.

In other words, this feature allows output records to be *multiplexed* to different target files depending on the values of the first fields in the output record.

To make an example, let's assume that we want to query the Windows Event Log, and for each event source name, we want to create a \underline{CSV} text file containing all the distinct event ID's generated by that source name.

The command would look like the following example:

LogParser "SELECT DISTINCT SourceName, EventID INTO Event_*.csv F ROM System" -i:EVT -o:CSV

For each output record generated by this query, the "SourceName" field will be used to substitute the wildcard in the target filename, and the "EventID" field will be written to the CSV file with the resulting file name. After the command execution is complete, we will have as many CSV output files as the number of different event source names:

C:\>dir

Volume in drive C has no label.

Earchume & dila will mentain the cust in the event ID's generated by the event source:

Directory of C:

C:\>type Event_Tcpip.csv

05%/ethen/120004 08:56 AM <DIR>

Three is 0004 line is 0004 line number of wildcard characters that can be used in 13 Event_Application Popup.csv

We can modify the example above to generate a *directory* for each event

conten/2004heerEmAder of event330Egged_WithOthatshD:

33 Event Dhcp.csv 07/19/2004 08:56 AM

27/29/23094'SEEECANSourceName, Eventup, COPINS(*) AS Total INTO *\ID

07/28/20ROM8556tAMGROUP B27 Sound Tank Event D" -i:EVT -o:CSV ATTEL \$1/2000 not a factor is 2 of mplete, EMEP Will have as many d072/d10/12064a.98/fies nAUMber of difference veeve every e8004 sources names:

16 Event Kerberos.csv 07/19/2004 08:56 AM

15 Event_NETLOGON.csv 07/19/2004 08:56 AM

07619/2014 dog 56 Als no label. 15 Event_NtServicePack.csv

Earth Strees to By The Astra in 935-0420 ground strain str duffetenatoevent_sourceeAccess.csv

03/10/2004 f08:56 AM 14 Event SCardSvr.csv

39 Event_Service Control Manager.csv 07/19/2004099:56 AM

07/19/2004 d09:56 AM nor DBRD 21 Event_Tcpip.csv

Example Apple Added to the second to the sec

e07/19/2004 09:59 AM <DIR>14 Evento Ministressup

07/10/2004 00:00

07X10/2019C096091A MOO 202509tes DCOM

07/110/2004Dig 08341/140,7127148 bytepfree

For the support for a signation of the support the "multiplex"

fee#1972004 09:08 AM <DIR>10 IDE100002003V

- 07/19/2004 09:08 AM <DIR>10 IDGEOMIRCOSV
- 07/19/2004 09:08 AM <DIR> i8042prt
- 07/19/2004 09:08 AM <DIR> **Kerberos**
- •07/19/2004 09:08 AM <DIR> **NETLOGON**
- •0 1/13 2004 09:08 AM <DIR> **NtServicePack**
- •07/39/2004 09:08 AM <DIR> Print
- •07/19/2004 09:08 AM <DIR> **RemoteAccess**
- 07/19/2004 09:08 AM <DIR> SCardSvr
- 07/19/2004 09:08 AM <DIR> Service Control Manager 07/19/2004 09:08 AM crosoft Corporation. All rights reserved.
- 07/19/2004 09:08 AM <DIR> W32Time
- 07/19/2004 09:08 AM <DIR> Win32k
- 07/19/2004 09:08 AM <DIR> Workstation
 - 0 File(s) 0 bytes
 - 21 Dir(s) 34,340,712,448 bytes free

Converting File Formats

Converting a log file from one format to another can be easily accomplished with Log Parser by executing a command with the following characteristics:

- The input format chosen for the command should match the conversion source format;
- The output format chosen for the command should match the conversion target format;
- The query should contain a <u>SELECT clause</u> that performs the necessary modifications on the input format field names and values in order to match the requirements of the target format.

When using Log Parser to convert one log file format to another, we should pay close attention to the order and names of the fields in the input and output formats. Some output formats, such as the <u>IIS output</u> format, have fixed fields. When converting to IIS log format, input format fields should be selected to match the IIS format exactly. For example, when converting a <u>W3C Extended log file</u> to IIS log format, we should select the client IP address first, the user name next, and so on.

In addition, we might want to change the name of the fields that we extract from the input format. For example, when writing to a W3C Extended format log file, Log Parser retrieves the names to be written to the "#Fields" directive from the SELECT clause. If we retrieve data from an IIS log format file, these names are not the same as those used by the W3C Extended format, so we must *alias* every field in order to get the correct field name.

As an example, consider the following SELECT clause that converts IIS log format files to IIS W3C Extended log format:

SELECT TO_DATE(TO_UTCTIME(TO_TIMESTAMP(Date, Time))) AS dat e,

We can see that the individual fields travely an arenamed as ording to the VS2 Standard convention of that the output file is fully compliant with

theolds words stended tormate,

In Second it ibra, Sheip date" and "time" fields are converted from local time, which is used in the which is used in the VKSCP Excended Rola format, to UTC time, which is used in the VKSCP Excended Rola format, u0009\u000a\u000d', '+') AS cs-uri-stem,

Parameters AS cs-uri-query,

UserName AS cs-username,

The command-line Log Parser executable can be used to run built-in queries that perform conversions between the following formats: Win32StatusCode AS sc-win32-status,

- Byles \$@nW&S sc-bytes,
- Byget & As cs-bytes,
- Time Takes AS time-taken
- IISW3C to IIS

For more information, refer to the <u>Command-Line Operation</u> reference.

© 2004 Microsoft Corporation. All rights reserved.

Custom Plugins

Log Parser allows users to develop custom input formats and use them with both the command-line Log Parser executable and with the Log Parser scriptable COM components.

There is no requirement on the language that can be used to implement a custom input format; for example, custom input formats can be implemented using any of the following languages:

- C++
- C#
- Visual Basic®
- JScript® or VBScript

Custom input formats are developed as COM objects implementing the methods of the <u>ILogParserInputContext</u> COM interface. There are two ways to write a COM object that implements the methods of this interface: implementing the ILogParserInputContext interface directly, or implementing the **IDispatch** (Automation) interface exposing the methods of the ILogParserInputContext interface.

Implementing the ILogParserInputContext Interface Directly

With this method, a Log Parser custom input format COM object must implement the ILogParserInputContext interface directly. This method usually requires writing C++ or Visual Basic code.

Implementing the IDispatch Interface Exposing the ILogParserInputContext Interface Methods

With this method, a Log Parser custom input format COM object must implement the IDispatch interface, and support the same methods exposed by the ILogParserInputContext interface. This method usually requires writing *scriptlets* (.wsc) files in JScript or VBScript. COM input format plugins that implement the IDispatch interface can also support custom properties. Custom input format COM objects must be registered with the COM infrastructure in order to be accessible by Log Parser. This task can be usually achieved using the *regsvr32.exe* tool distributed with the Windows OS. The following command registers a custom input format COM object implemented as a dynamic link library (*dll*):

C:\>regsvr32 myinputformat.dll

The following command registers a custom input format COM object implemented as a *scriptlet* JScript or VBScript file:

C:\>regsvr32 myinputformat.wsc

Once developed and registered with the COM infrastructure, custom input formats can be used with either the command-line Log Parser executable, or with the Log Parser scriptable COM components.

Using Custom Input Formats with the Command-Line Log Parser Executable

With the command-line Log Parser executable, custom input formats are used through the <u>COM</u> input format, which allows users to specify the **ProgID** of the custom COM object and eventual run-time properties.

As an example, let's assume that we have just developed a custom input format, and that its **ProgID** is "MySample.MyInputFormat". With the COM input format, the custom COM object can be used as follows:

C:\>logparser "SELECT * FROM inputfile" -i:COM -iProgID:MySample.MyI nputFormat

In the example above, "inputfile" stands for the specific <u>from-entity</u> recognized by the custom input format.

If we implemented our COM object through an **Automation** interface, we could also have our object support custom *properties*, and set them through the COM input format as shown in the following example:

C:\>logparser "SELECT * FROM inputfile" -i:COM -iProgID:MySample.MyI nputFormat -iCOMParams:ExtendedFields=on

For more information on the COM input format, refer to the <u>COM Input</u> <u>Format</u> reference.

Using Custom Input Formats with the Log Parser Scriptable COM Components

With the Log Parser scriptable COM components, custom input format objects are passed as the *inputFormat* argument to the <u>Execute</u> or <u>ExecuteBatch</u> methods of the <u>LogQuery</u> object.

The following VBScript example shows how our "MySample.MyInputFormat" custom COM object can be used with the Log Parser scriptable COM components:

```
Dim oLogQuery
Dim oMyInputFormat
Fprimors information on the Log Parser scriptable COM components,
spinl of the Com API Overview, and COM API Reference.
```

Set oLogQuery = CreateObject("MSUtil.LogQuery")

```
' Create our custom Input Format object
Set oMyInputFormat = CreateObject("MySample.MyInputFormat")
```

```
' Create Output Format object
Set oCSVOutputFormat = CreateObject("MSUtil.LogQuery.CSVOutputForma
t")
oCSVOutputFormat.tabs = TRUE
```

```
' Create query text
strQuery = "SELECT TimeGenerated, EventID INTO C:\output.csv FROM Sy
stem"
strQuery = strQuery & "WHERE SourceName = 'Application Popup'"
```

```
'Execute query
oLogQuery.ExecuteBatch strQuery, oMyInputFormat, oCSVOutputFormat
```

Custom Input Format Samples

Log Parser comes with three custom input format samples, located in the "Samples\COM" folder:

- **Processes**: this sample shows how to write a custom input format using the C++ language;
- **BooksXML**: this sample shows how to write a custom input format that parses XML documents, using the C# language;
- **QFE**: this sample shows how to write a custom input format that returns information gathered through a WMI query, using the VBScript language.

For more information on custom input format plugins and the ILogParserInputContext interface, refer to the <u>COM Input Format Plugins</u> reference.

© 2004 Microsoft Corporation. All rights reserved.

Log Parser COM API Overview

The Log Parser scriptable COM components offer numerous advantages and more flexibility than the command-line executable binary. For example, with the Log Parser scriptable COM components we can execute a query without providing an output format, retrieve the result output records, and process the output records ourselves.

The Log Parser scriptable COM components are implemented as **Automation** objects, which means that they can be used from any programming environment supporting automation, including C++, C#, Visual Basic, JScript and VBScript.

☑ Tip: Before using the Log Parser scriptable COM components on a computer, the "LogParser.dll" binary should be *registered* with the computer's COM infrastructure by executing the following command in the directory containing the "LogParser.dll" binary: C:\LogParser>regsvr32 LogParser.dll

The Log Parser scriptable COM components architecture is made up of the following objects:

- <u>MSUtil.LogQuery object</u>: this is the main COM object in the Log Parser scriptable COM components architecture; it exposes the main API methods and provides access to other objects in the architecture.
- <u>Input Format objects</u>: these objects provide programmatic access to the input formats supported by Log Parser; each input format object exposes properties having the same name as the parameters of the corresponding Log Parser input format.
- <u>Output Format objects</u>: these objects provide programmatic access to the output formats supported by Log Parser; each output format object exposes properties having the same name as the parameters of the corresponding Log Parser output format.

When writing an application that uses the Log Parser scriptable COM components, the very first step should be the instantiation of the **MSUtil.LogQuery** COM object.

The following JScript example shows how the MSUtil.LogQuery object is

instantiated by a JScript application:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

The following VBScript example shows how the MSUtil.LogQuery object is instantiated by a VBScript application:

Dim oLogQuery

Set oLogQuery = CreateObject("MSUtil.LogQuery") Once the **MSUtil.LogQuery** COM object has been instantiated, an application would usually proceed by executing a query in either *batch mode* or *interactive mode*, depending on the task that needs to be accomplished.

Batch Mode

A query executed in batch mode will have its output records written directly to an output format.

Batch mode works in the same way as the commands used with the Log Parser command-line executable, and it is useful when we want to execute a query and have its results sent to an output format, with no application intervention on the query output records.

A query is executed in batch mode by calling the <u>ExecuteBatch</u> method of the **MSUtil.LogQuery** object. This method takes three arguments:

- The text of the SQL-Like query;
- An input format object;
- An output format object.

The basic steps of an application using batch mode resemble the commands used with the Log Parser command-line executable:

- 1. Instantiate the MSUtil.LogQuery object;
- 2. Instantiate the input format object corresponding to the input format chosen for the query;
- 3. If needed, set input format object properties to change the default behavior of the input format;
- 4. Instantiate the output format object corresponding to the output format chosen for the query;
- 5. If needed, set output format object properties to change the default behavior of the output format;
- 6. Call the <u>ExecuteBatch</u> method of the **MSUtil.LogQuery** object, specifying the query text, the input format object, and the output format object.

The following examples show a simple application that creates a CSV file

containing selected records from the event log.

After instantiating the main **MSUtil.LogQuery** object, the application instantiates the <u>MSUtil.EVTInputFormat</u> input format object, which implements the <u>EVT</u> input format, and sets its <u>direction</u> property to "BW", in order to read events from the latest to the earliest. Then, the application instantiates the <u>MSUtil.CSVOutputFormat</u> output format object, which implements the <u>CSV</u> output format, and sets its <u>tabs</u> property to "ON", in order to improve readability of the CSV file. Finally, the application calls the <u>ExecuteBatch</u> method of the **MSUtil.LogQuery** object, specifying the query, the input format object, and the output format object; the method will execute the query, reading from the event log and writing to the specified CSV file, and will return

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

VBS EFine AXAMPLE in at object

var oEVTInputFormat = new ActiveXObject("MSUtil.LogQuery.EventLogInp DtFormhatg)Query

DEN DEN UtFrom Eachirection = "BW";

when the guery execution is complete.

Dim oCSVOutputFormat

Different Querty ut Format object

```
var oCSVOutputFormat = new ActiveXObject("MSUtil.LogQuery.CSVOutput
Setmad'gQuery = CreateObject("MSUtil.LogQuery")
```

```
oCSVOutputFormat.tabs = true;
```

' Create Input Format object

```
Set relate T Impnyt Ferrin at = CreateObject("MSUtil.LogQuery.EventLogInputFor
marts)rQuery = "SELECT TimeGenerated, EventID INTO C:\\output.csv FRO
ME SystepultFormat.direction = "BW"
```

strQuery += "WHERE SourceName = 'Application Popup'';

' Create Output Format object

Set xecSiteQutpytFormat = CreateObject("MSUtil.LogQuery.CSVOutputFormat b)ogQuery.ExecuteBatch(strQuery, oEVTInputFormat, oCSVOutputFormat) oCSVOutputFormat.tabs = TRUE ' Create query text strQuery = "SELECT TimeGenerated, EventID INTO C:\output.csv FROM Sy stem" strQuery = strQuery & "WHERE SourceName = 'Application Popup'"

' Execute query

oLogQuery.ExecuteBatch strQuery, oEVTInputFormat, oCSVOutputFormat

Interactive Mode

Queries executed in interactive mode do not use output formats, but rather return their output records directly to the application. Interactive mode is useful when we want to execute a query and receive the output records for custom processing.

A query is executed in interactive mode by calling the <u>Execute</u> method of the **MSUtil.LogQuery** object. This method takes two arguments:

- The text of the SQL-Like query;
- An input format object.

The **Execute** method returns a LogRecordSet object. The **LogRecordSet** object is an enumerator of LogRecord objects; it allows an application to navigate through the query output records. Each **LogRecord** object represents a single query output record, and it exposes methods that can be used to retrieve individual field values from the output record.

The basic steps of an application using interactive mode are:

- 1. Instantiate the MSUtil.LogQuery object;
- 2. Instantiate the input format object corresponding to the input format chosen for the query;
- 3. If needed, set input format object properties to change the default behavior of the input format;
- 4. Call the <u>Execute</u> method of the **MSUtil.LogQuery** object, specifying the query text and the input format object, and receiving a **LogRecordSet** object;
- Enter a loop that uses the <u>atEnd</u>, <u>getRecord</u>, and <u>moveNext</u> methods of the LogRecordSet object to enumerate the LogRecord query result objects;
- 6. For each **LogRecord** object, access its field values using the <u>getValue</u> method of the **LogRecord** object, and process the

field values as needed;

7. When finished, dispose of the **LogRecordSet** object by calling its <u>close</u> method.

The following examples show a simple application parsing an IIS web site's logs and printing the output records to the console output. After instantiating the main **MSUtil.LogQuery** object, the application instantiates the <u>MSUtil.IISW3CInputFormat</u> input format object, which implements the <u>IISW3C</u> input format.

Then, the application calls the <u>Execute</u> method of the **MSUtil.LogQuery** object, specifying the query and the input format object, and receiving the resulting **LogRecordSet** object.

The **LogRecordSet** object is used in a loop to enumerate the **LogRecord** objects implementing the query output records; the application retrieves the first field from each **LogRecord** object and prints it to the console output.

Finally, the application disposes of the **LogRecordSet** object by calling its <u>close</u> method.

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

var oIISW3CInputFormat = new ActiveXObject("MSUtil.LogQuery.IISW3CI
fpintFormatCipery

Dim oIISW3CInputFormat

DimestrQqueey_text DimestrQqueey_text DimstrQueewydSeSELECT c-ip FROM <1> WHERE cs-uri-stem LIKE '%hitco DimstrClientIp // Execute query and receive a LogRecordSet Set oRegQdSet = GreegQUbjycE(XMGte(tistEQgQyeryII)\$W3CInputFormat);

%/Cv/esitealhpettdrdsmat object

Settile(ISVR@CdmpSeeF.etFind())CreateObject("MSUtil.LogQuery.IISW3CInputFo

t(mat")

// Get a record

```
'CreaterqnRegutedt⊨ oRecordSet.getRecord();
```

strQuery = "SELECT c-ip FROM <1> WHERE cs-uri-stem LIKE '%hitcount.a
sp''' // Get first field value

```
var strClientIp = oRecord.getValue( 0 );
```

' Execute query and receive a LogRecordSet

```
Set oR & rond Sield vallog Query. Execute (strQuery, oIISW3CInputFormat)
```

```
WScript.Echo( "Client IP Address: " + strClientIp );
```

' Visit all records

DO WHAIdstante @ LogReccord Set tate net xt record

oRecordSet.moveNext();

} 'Get a record

```
Set oRecord = oRecordSet.getRecord
```

```
// Close LogRecordSet
```

```
oRec'ordes firstosie(); value
```

```
strClientIp = oRecord.getValue ( 0 )
```

' Print field value WScript.Echo "Client IP Address: " & strClientIp

' Advance LogRecordSet to next record oRecordSet.moveNext

LOOP

' Close RecordSet oRecordSet.close

C# Example

The Log Parser scriptable COM components can be easily consumed by .NET applications using the COM interop feature of the .NET Framework.

The COM interop feature of the .NET framework allows users to instantiate and use COM objects through the use of Runtime Callable Wrappers (RCW).

The RCW is a .NET class that wraps a COM object and gives a .NET application the notion that it's interacting with a managed .NET component.

RCW's are created by either using the *Type Library Importer* (tlbimp.exe) tool, or by importing a reference to the Log Parser scriptable COM objects through the Microsoft Visual Studio® .NET user interface. In either case, the RCW's are generated and stored in an assembly named "Interop.MSUtil.dll", which contains Runtime Callable Wrappers for all of the Log Parser scriptable COM components. By referencing this assembly, our .NET applications can use the Log Parser scriptable COM components as if they were managed .NET components.

The following example C# application executes a Log Parser guery that returns the latest 50 events from the System event log, printing the guery results to the console output:

using System;

{

```
using LogQuery = Interop.MSUtil.LogQueryClassClass;
```

Thenfollowing steps describe have to his difficult the semple tepplication intext Cla ssClass;1. Build an interop assembly containing the Runtime Callable

using LogRecordSet = Interop MSUtil IL ogRecordset; COM components.

This step, can by executed in two different ways:

- class LogParserSample From within a Visual Studio .NET project, *import a reference* to the Log Parser scriptable COM components; public static void Main(string[] Args) • From a command-line shell, execute the tlbimp.exe tool

 - (generally available in the "Bin" folder of the .NET try
 - framework SDK), specifying the path to the LogParser.dll {
 - // Installiate the LogQuery object

LogQuery Suban LogParser. LogQuery Store Description of the second secon

 Compile the sample source file into an executable, referencing the newly created "Interop.MSUtil.dll" assembly.
 // Set its "direction" parameter to "BW" From a command-line shell, this step can be executed as oEVTI put Format.direction = "BW";

// Create the query C:\>csc /r:Interop.MSUtil.dll /out:Events.exe sample.cs string query = @"SELECT TOP 50 SourceName, EventID, Message F ROM System";

© 2004 Microsoft Corporation. All rights reserved.

// Execute the query

();

}

}

LogRecordSet oRecordSet = oLogQuery.Execute(query, oEVTInputFo
rmat);

```
// Browse the recordset
for(; !oRecordSet.atEnd(); oRecordSet.moveNext())
{
    Console.WriteLine(oRecordSet.getRecord().toNativeString(","));
}
// Close the recordset
    oRecordSet.close();
}
catch(System.Runtime.InteropServices.COMException exc)
{
    Console.WriteLine("Unexpected error: " + exc.Message);
}
```

Security Considerations

• When using input and output formats to retrieve and send data over the network, users should be aware that most of the protocols utilized for data transfer (e.g. SMB, HTTP, and SYSLOG) do not make use of encryption, and could thus be vulnerable to interception and tampering by malicious entities.

In order to provide a secure environment in which these network connections are less vulnerable to interception, users should implement the IPSec protocol on their networks, and/or use SSL HTTP connections when retrieving data from a Web URL.

- When using the <u>Incremental Parsing</u> feature, users should store their checkpoint files in a secure location, and verify that checkpoint files have proper ACL's (Access Control Lists) preventing malicious entities from tampering with the data that the Log Parser input formats store in the checkpoint files.
- When implementing <u>custom input format COM objects</u>, users should ensure that the objects are not accessible from local and remote lowprivileged users, in order to prevent malicious entities from instantiating and using the custom input format objects from the local computer or from a remote computer.

In order to deny access to low-privileged users, either set proper ACL's on the custom input format COM objects' binaries, or use the "DCOM Configuration" Management Console (available in the "Administrative Tools" folder under the "Component Services" management console) to explicitly allow selected users only local access to your custom input format COM objects.

When using the <u>SQL output format</u>, users should be aware that the ODBC connection properties provided through the <u>SQL output format</u> parameters, which include username and password, could be transmitted over the network in clear text. In addition, the data transmitted through the ODBC connection could be unencrypted and thus vulnerable to interception and tampering by malicious entities. In order to provide a more secure environment, users should create a Data Source Name (DSN) on the local computer specifying the connection properties to use for the connection to the database, and

specify the name of the Data Source as a value to the <u>dsn</u> parameter of the SQL output format. Using a Data Source Name for the connection provides the following benefits:

- The username and password for the connection are stored securely by the ODBC subsystem;
- Certain ODBC drivers, including Microsoft SQL ServerTM ODBC drivers and Microsoft Access ODBC drivers, provide an option that allows users to enable encryption of the network traffic between the ODBC connection endpoints.

For more information on securing the communication between the ODBC connections endpoints, see the MSDN® <u>Data Access Security</u> topic.

• When processing sensitive or confidential data, users should provide proper ACL's on the files generated by the output formats or on the directories in which the output formats generate files, in order to prevent malicious entities from accessing and/or tampering with the output data generated by a query.

© 2004 Microsoft Corporation. All rights reserved.

Frequently Asked Questions

- 1. How do I specify yesterday's date?
- 2. <u>How do I retrieve the event logs that have been logged in the past 10 minutes?</u>
- 3. <u>After parsing my IIS log files, I get a message saying "There have been 4 parse errors." What causes this?</u>
- 4. How do I change the column names in my output file?
- 5. <u>How do I combine the IISW3C "date" and "time" fields into a</u> <u>single TIMESTAMP field?</u>
- 6. <u>How do I split a single TIMESTAMP field into a date-only field</u> <u>and a time-only field?</u>
- 7. When I use a "SELECT *" on an IIS W3C Extended log file, I get many fields with NULL values. What causes this?
- 8. <u>I get an error saying "Unknown field XYZ" when I execute my query. How do I fix this?</u>
- 9. <u>I am trying to write a query that uses the IN operator, but Log</u> <u>Parser keeps giving me errors. What am I doing wrong?</u>
- 10. When I execute a "SELECT *" on a log file, the output records contain 2 extra fields that I can not find in the log. What are these fields?
- 11. <u>I am developing an ASP or ASP.Net or Scheduled Task</u> application with Log Parser, and I'm having problems with permissions. What can I do?
- 12. <u>Can I use the Log Parser scriptable COM components from a</u> <u>multi-threaded application?</u>

How do I specify yesterday's date?

You need to use the <u>SUB</u> function to subtract one day from the current UTC timestamp returned by the <u>SYSTEM_TIMESTAMP</u> function.

The origin for TIMESTAMP values is January 1, year 0 at 00:00:00. This means that a time span of one day is represented by the timestamp for January 2, year 0 at 00:00:00, i.e. 24 hours after the origin of time.

Use the following field-expression to specify yesterday's date:

SUB (SYSTEM_TIMESTAMP(), TIMESTAMP('01-02', 'MM-dd'))

For more information, see the <u>TIMESTAMP Reference</u>.

How do I retrieve the event logs that have been logged in the past 10 minutes?

You need to use the <u>SUB</u> function to subtract 10 minutes from the current UTC timestamp returned by the <u>SYSTEM_TIMESTAMP</u> function, and convert this timestamp to local time using the <u>TO_LOCALTIME</u> function:

SELECT *

FROM System

WHERE TimeGenerated >= TO_LOCALTIME(SUB(SYSTEM_TIMES After parsing $m_{M}HS_{I}$ or files, I get a message saying "There have been 4 parse errors." What causes this?

Your log files are somehow malformed. This might happen, for example, if a client requests a URL or specifies a user name containing spaces. Log Parser cannot process that row and skips it. To see exactly what's going on, set the <u>-e global switch</u> to any value greater than or equal to zero. This makes Log Parser stop the query execution when that number of parse errors is encountered, and dump all the messages of the parse errors that occurred. For more information, see <u>Errors, Parse Errors, and Warnings</u>.

How do I change the column names in my output file?

Use the AS keyword in your <u>SELECT</u> clause to *alias* the field. For example:

SELECT Field1 AS newFieldName, Field2 AS newFieldName2, ...

How do I combine the IISW3C "date" and "time" fields into a single TIMESTAMP field?

Use the <u>TO_TIMESTAMP</u> function, as in the following example:

SELECT TO_TIMESTAMP(date, time), ...

How do I split a single TIMESTAMP field into a date-only field and a time-only field?

Use the <u>TO_DATE</u> and <u>TO_TIME</u> functions, as in the following example:

SELECT TO_DATE(myTimestamp), TO_TIME(myTimestamp), ...

For more information, see the <u>TIMESTAMP Reference</u>.

When I use a "SELECT *" on an IIS W3C Extended log file, I get many fields with NULL values. What causes this?

The <u>IISW3C</u> input format has 32 fields, which are all the possible fields that IIS 5.0 and IIS 6.0 can log. If your Web Server is configured to log only a few of these fields, the IISW3C input format returns the other field values as NULL values.

I get an error saying "Unknown field XYZ" when I execute my query. How do I fix this?

If you have not specified an input format for your query, Log Parser chooses one automatically based on the <from-entity> in the FROM clause of your query. In some cases, the input format might not be the one you expect.

Try specifying the input format explicitly using the <u>-i switch</u>. If you have specified the correct input format, make sure that you have typed the field name correctly.

I am trying to write a query that uses the IN operator, but Log Parser keeps giving me errors. What am I doing wrong?

Make sure you are separating the values on the right-side of the IN operator with the correct separator.

If the IN operator is comparing a single field-expression with a list of values, separate the values with a semicolon (;), not with a comma, as follows:

WHERE MyField IN ('VALUE1'; 'VALUE2'; 'VALUE3')

Different values for the same field-expression ("value-rows") are separated by a semicolon; comma characters are used to separate values within a single value-row.

For more information, see the <u>IN Operator Reference</u>.

When I execute a "SELECT *" on a log file, the output records contain 2 extra fields that I can not find in the log. What are these fields?

Most of the input formats add some tracking fields to the input records, such as the name of the file currently parsed, and the row number currently parsed.

If you do not want these fields to appear in your output records, do not use "SELECT *". Instead, specify only the field names that you want, as in the following example:

SELECT Field1, Field2, Field3,

I am developing an ASP or ASP.Net or Scheduled Task application with Log Parser, and I'm having problems with permissions. What can I do?

The first step in troubleshooting these problems is identifying the account under which Log Parser is running. If you are developing an

ASP or ASP.Net application, Log Parser will run as the account of the user requesting the page. If the request is anonymous, the account is the IIS Anonymous account; if the request is authenticated, the account is the authenticated user's account. If you are developing a Scheduled Task application, the account is the account that you have specified for the task.

Once the account has been identified, appropriate permissions must be given for this account to access both the Log Parser binary and the Dynamic Link Libraries that Log Parser depends to, which include standard Windows libraries (e.g. "kernel32.dll", "user32.dll", etc.) and a significant number of other libraries (e.g. "WinInet.dll", "odbcint.dll", etc.).

Finally, appropriate permissions must be given for the account to access the data that your application asks Log Parser to process. These may include IIS log files, the Event Log, text files, and whatever data you are processing.

Note: It is **not** a good security practice to change system ACL's and permissions to grant user accounts access to protected system resources. This is especially true if you are developing an external-facing web application that uses Log Parser to display information to the users. In these cases, consider instead developing a Scheduled Task that runs under a "private" account, and that generates at frequent intervals the web pages that your application will display to the user.

Can I use the Log Parser scriptable COM components from a multithreaded application?

The Log Parser scriptable COM components are registered to run within a single-threaded COM apartment, meaning that the objects *can* be used from multiple threads, but calls to the objects' methods will be serialized by the COM infrastructure to guarantee that only one thread at a time can access the components.

Query Syntax

<query> ::= <select_clause>[<using_clause>]
 [<into_clause>]
 <from_clause>]
 [<where_clause>]
 [<group_by_clause>]
 [<having_clause>]
 [<order_by_clause>]

Remarks:

 A query can include *comments*, that is, user-provided text not evaluated by Log Parser, used to document code or temporarily disable parts of query statements.
 For more information, read the <u>Comments Reference</u>.

Examples:

A. Minimal query

The following example shows the *minimal* query that can be written with the Log Parser SQL-Like language, making use of the SELECT and FROM clauses only:

SELECT TimeGenerated, SourceName

FROM System

B. Complete query

The following example shows a complete query that makes use of all the clauses in the Log Parser SQL-Like language:

SELECT TypeName, COUNT(*) AS TotalCount USING TO_UPPERCASE(EXTRACT_TOKEN(EventTypeName, 0, ' ')) AS TypeName INTO Report.csv SeeFelSM System SELVELERE TypeName LIKE '%service%' USMROUP BY TypeName INTEAVING TotalCount > 5 FROM DER BY TotalCount DESC WHERE GROUP BY HAVING ORDER BY

<u>Comments</u>

© 2004 Microsoft Corporation. All rights reserved.

SELECT

<select_clause></select_clause>	::=	<pre>SELECT [TOP <integer>] [DISTINCT ALL] <selection_list></selection_list></integer></pre>
<selection_list></selection_list>	::=	<selection_list_el> [, <selection_list_el>]</selection_list_el></selection_list_el>
<selection_list_el></selection_list_el>	::=	<field_expr> [AS <alias>] *</alias></field_expr>

The SELECT clause specifies the fields of the output records to be returned by the query.

Arguments:

TOP n

Specifies that only the first n records are to be output from the query result set. If the query includes an <u>ORDER BY clause</u>, the first n records ordered by the ORDER BY clause are output. If the query has no ORDER BY clause, the order of the records is arbitrary. For more information, see <u>Retrieving a Fixed Number of Records</u>.

ALL

Specifies that duplicate records can appear in the result set. ALL is the default.

DISTINCT

Specifies that only unique records can appear in the result set. NULL values are considered equal for the purposes of the DISTINCT keyword.

For more information, see <u>Eliminating Duplicate Values</u>.

<selection_list>

The fields to be selected for the result set. The selection list is a series of <u>field-expressions</u> separated by commas.

*

Specifies that all the input record fields should be returned. The fields are returned in the order in which they are exported by the Input Format.

AS <alias>

Specifies an alternative name to replace the field name in the query result set. By default, output formats that display field names use the text of a <u>field-expression</u> in the SELECT clause as the name of the corresponding output record field. However, when a field-expression in the SELECT clause has been aliased, output formats will use the alias as the name of the output record field.

The <u>alias</u> of a field-expression can be also used anywhere else in the query as a shortcut that refers to the original field-expression.

Remarks:

• When a field-expression is aliased with an alias matching an input record field name, the aliasing will affect that field-expression only; any other occurrence of the alias in the query will resolve to the input record field name.

As an example, the output records of the following query are made up of two fields with an identical name ("TimeGenerated"); the first output record field will contain values from the aliased field-expression ("ADD(EventID, 1000)"), while the second output record field will contain values from the "TimeGenerated" input format field:

SELECT ADD(EventID, 1000) AS TimeGenerated, TimeGenerated FROM system

• A field-expression in the SELECT clause can refer to aliases defined elsewhere in the SELECT clause, as long as the definition happens

before (in a left-to-right order) its use. The following example is a correct SELECT clause:

SELECT EventID AS MyAlias, ADD(MyAlias, 100)

On the other hand, the following example is not a correct SELECT clause, since the "MyAlias" alias is used before being defined:

SELECT ADD(MyAlias, 100), EventID AS MyAlias

Examples:

A. Selecting specific fields

The following query selects a subset of all the fields exported by the \underline{EVT} Input Format:

SELECT TimeGenerated, SourceName

FROM System

B. Selecting specific fields and <u>field-expressions</u>

The following query selects a <u>constant</u> and a <u>function</u> that uses a field exported by the <u>EVT</u> Input Format as argument:

SELECT 'Event Type:', EXTRACT_TOKEN(EventTypeName, 0, ' ') FROM System

C. Selecting all fields with *

The following query selects all the fields exported by the \underline{EVT} Input Format:

```
SELECT *
```

FROM System

D. Using TOP

The following query returns the 10 most requested Url's in the specified <u>IISW3C</u> log file:

SELECT TOP 10 cs-uri-stem, COUNT(*)

FROM ex040305.log

EGBOID BYSCEINCE

The following query uses the <u>REG</u> Input Format to return all the registry key value types that are found under the specified key:

SELECT DISTINCT ValueType

FROM \HKLM\SYSTEM\CurrentControlSet

F. Aliasing field-expressions

The following query returns a breakdown of page requests per page type from the specified <u>IISW3C</u> log file:

SELECT TO_UPPERCASE(EXTRACT_EXTENSION(cs-uri-stem)) AS PageType, COUNT(*) AS TotalHits FROM ex040305.log GROUP BY PageType Second Sort BY TotalHits DESC

Field Expressions Field Names and Aliases USING

Basics of a Query Eliminating Duplicate Values Retrieving a Fixed Number of Records

© 2004 Microsoft Corporation. All rights reserved.

USING

```
<using_clause> ::= USING <field_expr> AS <alias> [ , <field_expr> AS <alias> ... ]
```

The USING clause declares aliased field-expressions that do not appear in the output records but can be referenced anywhere in the query. The USING clause is employed to <u>improve query readability</u>.

Remarks:

• For more information on aliasing field-expressions, see the <u>SELECT</u> <u>Clause Reference</u>.

Examples:

A. Declaring aliased field-expressions

The following example query returns the "account name" portion of the fully-qualified account name that appears in the resolved "SID" field of the \underline{EVT} input format:

```
SELECT Username
USING TO_LOWERCASE( RESOLVE_SID(Sid) ) AS FQAccount,
EXTRACT_TOKEN( FQAccount, 1, '\\') AS Username
FROM Security
See also:
```

Field Expressions Field Names and Aliases SELECT

Improving Query Readability

© 2004 Microsoft Corporation. All rights reserved.

INTO

<into_clause> ::= INTO <into_entity>

The INTO clause is used to specify the output format target(s) to which the query output records are to be written.

Remarks:

- The syntax and interpretation of the <into_entity> specified in the INTO clause depends on the output format used. For information on the syntax and interpretation of the <into_entity> values supported by each output format, refer to the <u>Output Formats Reference</u>.
- Regardless of the output format used, the <into_entity> specified in the INTO clause must comply with the following general syntax:
 - The <into_entity> can not contain spaces, unless it is enclosed by the " (single quote) or " (double quotes) characters, as in the following example:

```
'C:\Program Files\file3.txt'
```

• The following characters are considered *parenthesys* characters, and if they appear in an <into_entity>, they must appear as well-formed pairs of *opening* and *closing* parenthesys:

<>()[]{}

The following examples show valid into-entities containing *parenthesys* characters:

```
entity<value>
```

```
entity[value]value
```

The following examples show invalid into-entities containing
parenthesys characters:

entity>value<

entity}value

Anyerhar(acter (including illegal characters and non-printable characters) in an <into-entity> can be entered using the **\uxxxx** notation, where *xxxx* is the 4-digit hexadecimal representation of the UNICODE character, as in the following example:

C:\Program\u0020Files\file3.txt

• Into-entities that represent names of files or directories are not allowed to contain the following characters, even when enclosed in quote characters or entered using the **\uxxxx** notation:

tab carriage-return line-feed , () " < >

• Since the INTO clause is not a mandatory clause in the Log Parser SQL-Like language, most output formats employ default <into_entity> values that are implicitly used when a query does not include an INTO clause.

For example, the <u>NAT</u>, <u>CSV</u>, and <u>TSV</u> output formats assume STDOUT when an INTO clause is not specified. For more information on the default <into_entity> values assumed by each output format, refer to the <u>Output Formats Reference</u>.

• The **TO** clause used by earlier versions of Log Parser has been deprecated in favor of the INTO clause.

Examples:

A. Explicit <into_entity>

The following example query specifies an explicit target CSV file for the \underline{CSV} output format:

SELECT *

BNmplicitOsiptocentity>

FROM System The following example query uses an implicit STDOUT target for the <u>NAT</u> output format:

SELECT *

FROM System

C. Explicit <into_entity>

The following example query specifies an explicit STDOUT target for the $\underline{\text{NAT}}$ output format:

SELECT *

INTO STDOUT FROM System

See also:

FROM

Basics of a Query Output Formats Reference

FROM

<from_clause> ::= FROM <from_entity>

The FROM clause is used to specify the input format source(s) from which the query input records are to be read.

Remarks:

- The syntax and interpretation of the <from_entity> specified in the FROM clause depends on the input format used. For information on the syntax and interpretation of the <from_entity> values supported by each input format, refer to the <u>Input Formats Reference</u>.
- Regardless of the input format used, the <from_entity> specified in the FROM clause must comply with the following general syntax:
 - The <from_entity> must be a single *element* or a *list* of elements, separated by the ',' (comma) or ';' (semicolon) characters, as in the following examples:

```
file1.txt
```

file1.txt, file2.txt

Each elementican not contain spaces, ',' (comma) characters, or ';' (semicolon) characters, unless the element is enclosed by the ''' (single quote) or ''' (double quotes) characters, as in the following example:

file2.txt, 'C:\Program Files\file3.txt', file4.txt

• The following characters are considered *parenthesys* characters, and if they appear in an *element*, they must appear as well-formed pairs of *opening* and *closing* parenthesys:

<>()[]{}

The following examples show valid from-entities containing *parenthesys* characters:

entity<value>

entity[value]value The following examples show invalid from-entities containing *parenthesys* characters:

entity>value<

entity}value

 Any characters (including illegal characters and non-printable characters) in a <from-entity> can be entered using the **\uxxxx** notation, where *xxxx* is the 4-digit hexadecimal representation of the UNICODE character, as in the following example:

 $C:\label{eq:constraint} C:\label{eq:constraint} C:\l$

• From-entities that represent names of files or directories are not allowed to contain the following characters, even when enclosed in quote characters or entered using the **\uxxxx** notation:

tab carriage-return line-feed , () " < >

Examples:

A. <from_entity> with the <u>**REG</u>** input format</u>

The following example query reads input records from the registry using the $\underline{\mathsf{REG}}$ input format:

```
SELECT *
```

FROM \HKLM\SOFTWARE

B. <from_entity> with the **EVT** input format

The following example query reads input records from the System and Security event logs using the \underline{EVT} input format:

SELECT * FROM System, Security

See also:

<u>INTO</u>

Basics of a Query Input Formats Reference

WHERE

<where_clause> ::= WHERE <expression>

The WHERE clause is used to specify a boolean condition that must be satisfied by an input record for that record to be output. Input records that do not satisfy the condition are discarded.

Remarks:

• The *expression* in a WHERE clause can not reference <u>SQL</u> (aggregate) functions. To specify conditions on values of aggregate functions, use the <u>HAVING</u> clause.

Examples:

A. Simple expression

WHERE EventID = 501

B. Complex expression

```
WHERE EXTRACT_TOKEN(Strings, 1, '|') LIKE '%logon&' AND
( TimeGenerated > SUB( TO_LOCALTIME(SYSTEM_TIMESTA
MP()), TIMESTAMP( '10', 'mm' ) ) OR
SID IS NOT NULL
See also:
```

Expressions HAVING

Filtering Input Records

GROUP BY

<group_by_clause></group_by_clause>	::=	GROUP BY <field_expr_list> [WITH</field_expr_list>
		ROLLUP]
<field_expr_list></field_expr_list>	::=	<field_expr> [, <field_expr>]</field_expr></field_expr>

The GROUP BY clause specifies the groups into which output rows are to be placed and, if <u>aggregate functions</u> are included in the <u>SELECT</u> or <u>HAVING</u> clauses, calculates the aggregate functions values for each group.

Arguments:

WITH ROLLUP

Specifies that in addition to the usual rows provided by GROUP BY, summary rows are introduced into the result set. Groups are summarized in a hierarchical order, from the lowest level in the group to the highest, and the corresponding summary rows contain NULL values for the groups that have been summarized.

The group hierarchy is determined by the order in which the grouping field-expressions are specified. Changing the order of the grouping field-expressions can affect the number of rows produced in the result set.

The ROLLUP operator is often used with the <u>GROUPING</u> aggregate function.

Remarks:

• When GROUP BY is specified, either each non-aggregate and nonconstant field-expression in the SELECT clause should be included in the GROUP BY field-expression list, or the GROUP BY fieldexpression list must match exactly the SELECT clause field-expression list. For more information, see <u>Aggregating Data Within Groups</u>.

- Aggregate functions using the DISTINCT keyword, for example, "COUNT(DISTINCT field-expression)", are not supported when using the GROUP BY clause.
- If the <u>ORDER BY</u> clause is not specified, groups returned using the GROUP BY clause are not in any particular order. It is recommended that the ORDER BY clause is always used to specify a particular ordering of the data.

Examples:

A. Simple GROUP BY clause

The following query, on an <u>IISW3C</u> log file, returns the number of requests for each page on each day:

SELECT date, cs-uri-stem, COUNT(*) FROM LogFiles\ex040528.log

The group & Demander that have been introduced by the rollup openator are/style.css 1

1

2003-11-18 /images/address.gif 1

2003-11-18 /cgi-bin/counts.exe 1

2003-11-19 /data/rulesinfg.nsf 2

V2003: 11200: 1200

2003-11-20 /top2.htm 1

2003-11-20 /top2.intil 1

- - 20

See also: 1-18 -	6
Field Expressions	6
Field Expressions SEI2003T 11-20 -	8

Aggregating Data Within Groups

HAVING

<having_clause> ::= HAVING <expression>

The HAVING clause is used to specify a boolean condition that must be satisfied by a group for the group record to be output. Groups that do not satisfy the condition are discarded.

Examples:

A. Simple expression

HAVING EventID = 501

B. Complex expression

HAVING SUM(sc-bytes) > 100000 AND

(COUNT(*) > 1000 OR

C. Complex/expressionSION(cs-uri-stem) LIKE 'htm'

The following example query retrieves all the event sources from the System event log that generated more than 10 events:

```
SELECT SourceName
FROM System
GROUP BY SourceName
HAVING COUNT(*) > 10
```

See also:

Expressions WHERE

Filtering Groups

ORDER BY

<field expr list>

<order_by_clause> ::= ORDER BY <field_expr_list> [ASC | DESC]

::=

<field expr> [, <field expr> ...]

The ORDER BY clause specifies which <u>SELECT</u> clause field-expressions the query output records should be sorted by.

Arguments:

ASC

Specifies that the field-expression list values should be sorted in ascending order, from lowest value to highest value. ASC is the default.

DESC

Specifies that the field-expression list values should be sorted in descending order, from highest value to lowest value.

Remarks:

- The Log Parser SQL-Like language requires that each field-expression appearing in the ORDER BY clause must also appear in the SELECT clause.
- Differently than the standard SQL language, in the Log Parser SQL-Like language the DESC or ASC sort direction applies to all the fieldexpressions in the ORDER BY clause. In other words, it is not possible to specify different sort directions for different field-expressions.
- NULL values are treated as the lowest possible values.

Examples:

A. Sorting by a single field-expression

SELECT date, cs-uri-stem, cs-uri-query, sc-bytes FROM LogFiles\ex040528.log BOBDETRB bycmyleipDE field-expressions

SELECT date, cs-uri-stem, cs-uri-query, sc-bytes FROM LogFiles\ex040528.log ORDER BY date, sc-bytes

See also:

Field Expressions SELECT

Sorting Output Records

Expressions

<expression></expression>	::=	<term1> [OR <expression>]</expression></term1>
<term1></term1>	::=	<term2> [AND <term1>]</term1></term2>
<term2></term2>	::=	<field_expr> <rel_op> <field_expr> <field_expr> [NOT] LIKE <like_mask> <field_expr> [NOT] BETWEEN <field_expr> AND <field_expr> <field_expr> IS [NOT] NULL <field_expr> [NOT] IN (<value_rows>) <field_expr> <rel_op> [ALL ANY] (<value_rows>) (<field_expr_list>) [NOT] IN (<value_rows>) (<field_expr_list>) <rel_op> [ALL ANY] (<value_rows>) NOT <term2> (<expression>)</expression></term2></value_rows></rel_op></field_expr_list></value_rows></field_expr_list></value_rows></rel_op></field_expr></value_rows></field_expr></field_expr></field_expr></field_expr></field_expr></like_mask></field_expr></field_expr></rel_op></field_expr>
<field_expr_list></field_expr_list>	::=	<field_expr> [, <field_expr>]</field_expr></field_expr>
<rel_op></rel_op>	::=	< > < > < = <= >= <= >= <
<value_rows></value_rows>	::=	<value_row> [; <value_row>]</value_row></value_row>

<value_row> ::= <u><value></u> [, <u><value></u> ...]

An expression is used in the <u>WHERE</u> and <u>HAVING</u> clauses to specify conditions that must be satisfied for input records or group records to be output.

Operators:

<rel_op>

Standard comparison operators (less than, greather than, etc.).

[NOT]LIKE

Indicates that the subsequent character string is to be used with pattern matching. For more information, see <u>LIKE</u>.

[NOT] BETWEEN

Specifies an inclusive range of values. Use AND to separate the beginning and ending values. For more information, see <u>BETWEEN</u>.

IS [NOT] NULL

The IS NULL and IS NOT NULL operators determine whether or not a given <u>field-expression</u> is NULL.

[NOT] IN

The IN and NOT IN operators determine whether or not a given <u>field-expression</u> or list of <u>field-expressions</u> matches any element in a list of values. For more information, see <u>IN</u>.

ALL

Used with a comparison operator and a list of values. Returns TRUE if all values in the list satisfy the comparison operation, or FALSE if

not all values satisfy the comparison. If no ALL nor ANY is specified, then ANY is assumed by default. For more information, see <u>ALL</u>.

ANY

Used with a comparison operator and a list of values. Returns TRUE if any value in the list satisfies the comparison operation, or FALSE if no values satisfy the comparison. If no ALL nor ANY is specified, then ANY is assumed by default. For more information, see <u>ANY</u>.

Remarks:

- The *expression* in a <u>WHERE</u> clause can not reference <u>SQL</u> (aggregate) functions. To specify conditions on values of aggregate functions, use the <u>HAVING</u> clause.
- There is no limit to the number of operators that can be included in an expression.
- The order of precedence for the logical operators is NOT (highest), followed by AND, followed by OR. The order of evaluation at the same precedence level is from left to right. Parentheses can be used to override this order in an expression.

Examples:

A. Simple expression

sc-bytes >= 1000

B. Complex expression

EXTRACT_TOKEN(Strings, 1, '|') LIKE '%logon&' AND

(TimeGenerated > SUB(TO_LOCALTIME(SYSTEM_TIMESTAMP()

), TIMESTAMP('10', 'mm')) OR SID IS NOT NULL

See) also:

ALL ANY BETWEEN IN LIKE

Constant Values Field Expressions HAVING WHERE

ALL

<field_expr> <rel_op> ALL (<value_rows>)

(<field_expr_list>) <rel_op> ALL (<value_rows>)

The ALL operator compares a given <u>field-expression</u> with a list of values, returning TRUE if all values in the list satisfy the comparison operation, or FALSE if not all values satisfy the comparison.

Examples

A. Single *field-expression*

The following example expression determines whether or not the "Year" field is greater than all the values in the specified list:

```
Year > ALL (1999; 2000; 2001)
```

B. List of field-expressions

The following example expression determines whether or not the pair of "Year" and "Age" fields is less than all the pairs of values in the specified list:

(Year, Age) < ALL (1999, 30; 2001, 40; 2002, 10)

See also:

ANY Expressions Field-Expressions

ANY

```
<field_expr> <rel_op> ANY ( <value_rows> )
```

(<field_expr_list>) <rel_op> ANY (<value_rows>)

The ANY operator compares a given <u>field-expression</u> with a list of values, returning TRUE if any value in the list satisfies the comparison operation, or FALSE if no values satisfy the comparison.

Examples

A. Single *field-expression*

The following example expression determines whether or not the "Year" field is greater than any value in the specified list:

```
Year > ANY (1999; 2000; 2001)
```

B. List of field-expressions

The following example expression determines whether or not the pair of "Year" and "Age" fields is less than any of the pairs of values in the specified list:

(Year, Age) < ANY (1999, 30; 2001, 40; 2002, 10)

See also:

ALL Expressions Field-Expressions

BETWEEN

<field_expr> [NOT] BETWEEN <field_expr> AND <field_expr>

The BETWEEN operator determines if a given <u>field-expression</u> belongs to a specified interval.

Examples

A. BETWEEN

The following example expression determines if the "Year" field belongs to the specified interval:

Year BETWEEN 1999 AND 2004

This example is equivalent to the following expression:

Year >= 1999 AND Year <= 2004

B. NOT BETWEEN

The following example expression determines if the "Year" field does not belong to the specified interval:

Year NOT BETWEEN 1999 AND 2004

This example is equivalent to the following expression:

Year < 1999 OR Year > 2004

C. TIMESTAMP interval

The following example query uses the <u>FS Input Format</u> to return all the files that have been created between 4 hours ago and 1 hour ago:

SELECT Path FROM C:\MyDir*.* WHERE TO_UTCTIME(CreationTime) BETWEEN SUB(SYSTEM_TI MESTAMP(), TIMESTAMP('4', 'h')) AND SUB(SYSTEM_TIMESTAM SepalsmmestamP('1', 'h'))

Expressions Field-Expressions

IN

```
<field_expr> [ NOT ] IN ( <value_rows> )
```

```
( <field_expr_list> ) [ NOT ] IN ( <value_rows> )
```

The IN and NOT IN operators determine whether or not a given <u>field-expression</u> or list of <u>field-expressions</u> matches any element in a list of values.

Remarks:

• Use the comma character (,) to separate values in a single list row, and use the semicolon character (;) to separate list rows.

Examples

A. Single *field-expression*

The following example expression determines if the "Age" field matches any value in the specified list:

```
Age IN (20; 30; 45; 60)
```

This example is equivalent to the following expression:

Age = 20 OR Age = 30 OR Age = 45 OR Age = 60

B. List of field-expressions

The following example expression determines if the pair of "FirstName" and "State" fields matches any pair of values in the specified list: (FirstName, State) IN ('Johnson', 'OR'; 'Smith', 'WA')

This example is equivalent to the following expression:

(FirstName = 'Johnson' AND State = 'OR') OR (FirstName = 'Smith' AN D State = 'WA')

See also:

Expressions Field-Expressions

LIKE

<field_expr> [NOT] LIKE <like_mask>

Determines whether or not a given character string matches a specified pattern. A pattern can include regular characters and wildcard characters. During pattern matching, regular characters must yield a case-insensitive match with the characters specified in the character string. Wildcard characters, however, can be matched with arbitrary fragments of the character string. Using wildcard characters makes the LIKE operator more flexible than using the = and != string comparison operators.

The wildcard characters that can be used in a LIKE pattern are:

• _ (*underscore* character): matches any single character Examples:

LIKE 'ab_d': matches all the four-letter strings that start with "ab" and end with "d" (e.g. "abcd", "AB+d")

LIKE 'a_c_': matches all the four-letter strings that have "a" in the first position and "c" in the third position (e.g. "abcd", "Akck")

• % (*percent* character): matches any string of zero or more characters Examples:

LIKE '%.asp' matches all the strings ending with ".asp" (e.g. "/default.asp", ".ASP")

LIKE '%error%' matches all the strings containing "error" (e.g. "an error has been found", "ERROR")

Remarks:

- Similarly to <u>STRING constants</u>, characters in a LIKE pattern can be escaped with the '\' (*backslash*) character or encoded with the **\uxxxx** notation.
- Wildcard pattern matching characters can be used as literal characters. To use a wildcard character as a literal character, escape the wildcard character with the '\' (*backslash*) character.

Examples:

```
LIKE 'ab\_d': matches the "ab_d" string (e.g. "ab_d", "AB_d")
LIKE 'a\%c%': matches all the strings that start with "a%c" (e.g. "a%cdefg", "A%c")
```

• When executing a Log Parser query from within a command-line batch file, using the % wildcard character might yeld unexpected results. For example, consider the following batch file:

@echo off

LogParser "SELECT * FROM SYSTEM WHERE Message LIKE '%ERRO Volven this batch file is executed, the command-line shell interpreter will assume that "%ERROR%" is a reference to an environment variable, and it will try to replace this string with the value of the environment variable. In most cases, such an environment variable will not exist, and the actual command executed by the shell will look like:

LogParser "SELECT * FROM SYSTEM WHERE Message LIKE ""

Which would yeld the following error:

Error: Syntax Error: <term2>: no valid LIKE mask

To avoid this problem, use double %% wildcard characters when writing a command-line batch file, as in the following example:

@echo off

LogParser "SELECT * FROM SYSTEM WHERE Message LIKE '%%ERR OR%%'"

Examples

A. LIKE

The following example WHERE clause finds all the URL's in an <u>IISW3C</u> log file that end with ".htm":

```
WHERE cs-uri-stem LIKE '%.htm'
```

B. NOT LIKE

The following example WHERE clause finds all the <u>Event Log</u> messages that do not contain "error":

WHERE Message NOT LIKE '%error%'

See also:

Expressions Field-Expressions

Field-Expressions

<field_expr> ::= <aggregate_function> <function> <field_name> <alias> <value>

Field-expressions are a combination of symbols and functions that Log Parser evaluates to obtain a single data value. These are the basic *arguments* of the <u>SELECT</u>, <u>USING</u>, <u>WHERE</u>, <u>GROUP BY</u>, <u>HAVING</u>, and <u>ORDER BY</u> clauses.

Field-expressions can be divided conceptually into two groups:

- *Derived* field-expressions: <u>functions</u> or <u>aggregate functions</u> having other field-expressions as arguments;
- Basic field-expressions: constant values (including functions with no arguments), names of input record fields, or aliases defined in the SELECT or USING clauses.

Examples:

A. Basic field-expressions

The SELECT clause in the following example query specifies "basic" field-expressions only:

```
SELECT 'Event ID:', EventID, SYSTEM_TIMESTAMP()
```

FROM System

B. Derived field-expressions

The SELECT clause in the following example query specifies "derived" field-expressions only:

SELECT TO_UPPERCASE(cs-uri-stem), SUM(sc-bytes) FROM \MyLogs\ex042805.log GROUP BY TO_UPPERCASE(cs-uri-stem)

See also:

Aggregate Functions Functions Constant Values Field Names and Aliases SELECT USING

Basics of a Query

Field Names and Aliases

<field_name> ::= [[] <string>[]] <alias> ::= [[] <string>[]]

Field names are names of fields of the input records generated by an input format.

Aliases are alternative names for <u>field-expressions</u>, assigned in the <u>SELECT</u> or <u>USING</u> clauses. When a field-expression in the SELECT clause has been aliased, output formats will use the alias as the name of the corresponding output record field.

The alias of a field-expression can be also used anywhere else in the query as a shortcut that refers to the original field-expression.

Remarks:

• The following characters are not allowed in field names or aliases, unless the field name or alias is enclosed in square brackets ([and]):

, ; < > = ! ' " @ * [] space

Field names and aliases containing spaces or illegal characters can be enclosed in square brackets ([and]), as in the following example:

SELECT [Last Request Time], [email@address], CPUTime as [Elapsed CP U Time]

 Arra of hapacters (including illegal characters and non-printable characters) inplied management aliases can be also entered using the luxxxx notation, where xxxx is the 4-digit hexadecimal representation of the UNICODE character:

SELECT Last\u0020Request\u0020Time FROM perflog.csv

- Field names and aliases can not match keywords or function names of the Log Parser SQL-Like language (e.g. "FROM", "ADD").
- Field names and aliases are not case-sensitive.

Examples:

A. Basic field-expressions

The SELECT clause in the following example query specifies "basic" field-expressions only:

SELECT 'Event ID:', EventID, SYSTEM_TIMESTAMP()

FROM System

B. Derived field-expressions

The SELECT clause in the following example query specifies "derived" field-expressions only:

SELECT TO_UPPERCASE(cs-uri-stem), SUM(sc-bytes) FROM \MyLogs\ex042805.log GROUP BY TO_UPPERCASE(cs-uri-stem)

See also:

<u>SELECT</u> <u>USING</u>

Basics of a Query

Aggregate Functions

<aggregate_function> ::= COUNT ([DISTINCT | ALL] *) COUNT ([DISTINCT | ALL] <field_expr_list>) SUM ([DISTINCT | ALL] <field_expr>) AVG ([DISTINCT | ALL] <field_expr>) MAX ([DISTINCT | ALL] <field_expr>) MIN ([DISTINCT | ALL] <field_expr>) PROPCOUNT (*) [ON (<on_field_expr_list>)] PROPCOUNT (<field_expr_list>) [ON (<on_field_expr_list>)] PROPSUM (<field_expr>) [ON (<on_field_expr_list>)] GROUPING (<field_expr>)

Aggregate functions perform a calculation on a set of values but return a single, summarizing value.

Aggregate functions are often used with the <u>GROUP BY</u> clause. When used without a GROUP BY clause, aggregate functions perform calculations on the entire set of input records, returning a single summarizing value for the whole set.

When used with a GROUP BY clause, aggregate functions perform calculations on each set of group records, returning a summarizing value for each group.

Functions:

COUNT

Returns the number of items in a group. For more information, see <u>COUNT</u>.

SUM

Returns the sum of the values of the specified field-expression. For more information, see \underline{SUM} .

AVG

Returns the average across the values of the specified fieldexpression. For more information, see AVG.

MAX

Returns the maximum value among the values of the specified fieldexpression.

For more information, see MAX.

MIN

Returns the minimum value among the values of the specified fieldexpression.

For more information, see MIN.

PROPCOUNT

Returns the ratio of the COUNT aggregate function calculated on a group to the COUNT aggregate function calculated on a hierarchically higher group.

For more information, see **PROPCOUNT**.

PROPSUM

Returns the ratio of the SUM aggregate function calculated on a group to the SUM aggregate function calculated on a hierarchically higher group.

For more information, see <u>PROPSUM</u>.

GROUPING

Returns a value of 1 when the row is added by the ROLLUP operator of the GROUP BY clause, or 0 when the row is not the result of ROLLUP.

The GROUPING aggregate function is allowed only when the GROUP BY clause contains the ROLLUP operator. For more information, see GROUPING.

Remarks:

- Aggregate functions are allowed as <u>field-expressions</u> only in the <u>SELECT</u>, <u>HAVING</u>, and <u>ORDER BY</u> clauses.
- The arguments of an aggregate function can not reference other aggregate functions.
- The arguments of an aggregate function can not reference the following <u>functions</u>:
 - <u>SEQUENCE</u>
 - <u>OUT_ROW_NUMBER</u>
- DISTINCT is allowed in aggregate functions only when there is no GROUP BY clause.

Examples:

A. COUNT(*)

The following query returns the total number of events in the System event log:

SELECT COUNT(*)

FROM System

B. COUNT(DISTINCT)

The following query returns the total number of distinct event source names in the System event log:
SELECT COUNT(DISTINCT SourceName)

FROM System

C. COUNT(*) and GROUP BY

The following query returns the total number of events generated by each event source in the System event log:

SELECT SourceName, COUNT(*)

FROM System

DG BONPARN GROUPABY

The following query returns the total number of bytes sent for each page extension logged in the specified <u>IIS W3C</u> log file:

SELECT TO_LOWERCASE(EXTRACT_EXTENSION(cs-uri-stem)) A

S PageType,

E. PROPC(OUNT(*), GROUP BY, and HAVING

FROM ex031118 log The following query returns the pages that represent more than 10% GROUP BY Page Type of the requests in the specified <u>IIS W3C</u> log file:

SELECT cs-uri-stem FROM ex031118.log GROUP BY cs-uri-stem HAVING PROPCOUNT(*) > 0.1 See also:

COUNT SUM AVG MAX MIN PROPCOUNT PROPSUM GROUPING

Functions SELECT HAVING <u>GROUP_BY</u>

Aggregating Data Within Groups Calculating Percentages

AVG

AVG ([DISTINCT | ALL] <field_expr>)

Returns the average among all the values, or only the DISTINCT values, of the specified <u>field-expression</u>.

Arguments:

DISTINCT

Specifies that AVG returns the average of unique values. DISTINCT can only be used when the query does not make use of the <u>GROUP BY</u> clause.

ALL

Applies the aggregate function to all values. ALL is the default.

<field_expr>

The <u>field-expression</u> whose values are to be averaged. The field-expression data type must be <u>INTEGER</u> or <u>REAL</u>.

Return Type:

INTEGER or **REAL**, depending on the argument field-expression.

Remarks:

- NULL values are ignored by the AVG aggregate function.
- Aggregate functions are allowed as <u>field-expressions</u> only in the <u>SELECT</u>, <u>HAVING</u>, and <u>ORDER BY</u> clauses.

- The arguments of an aggregate function can not reference other aggregate functions.
- The arguments of an aggregate function can not reference the following <u>functions</u>:
 - <u>SEQUENCE</u>
 - OUT_ROW_NUMBER
- DISTINCT is allowed in aggregate functions only when there is no GROUP BY clause.

Examples:

A. AVG

The following query returns the average number of bytes for executable files in the "system32" directory, using the FS input format:

SELECT AVG(Size)

FROM C:\windows\system32*.*

BWANG BATO GROUP BXSE(EXTRACT_EXTENSION(Name)) = 'exe'

The following query returns the average time spent by each page extension logged in the specified <u>IIS W3C</u> log file:

```
SELECT TO_LOWERCASE(EXTRACT_EXTENSION(cs-uri-stem)) A
S PageType,
AVG(time-taken)
FROM ex031118.log
Sec also IP BY PageType
COUNT
SUM
MAX
MIN
```

PROPCOUNT

PROPSUM

GROUPING

Aggregate Functions

Aggregating Data Within Groups

COUNT

COUNT ([DISTINCT | ALL] *) COUNT ([DISTINCT | ALL] <field_expr_list>)

<field_expr_list> ::= <field_expr> [, <field_expr> ...]

Returns the number of items in a group.

Arguments:

DISTINCT

Specifies that COUNT returns the number of unique values. DISTINCT can only be used when the query does not make use of the <u>GROUP BY</u> clause.

ALL

Applies the aggregate function to all values. ALL is the default.

*

Specifies that all records should be counted to return the total number of records, including records that contain NULL values.

<field_expr_list>

Specifies that only records for which at least one of the specified <u>field-expressions</u> is non-NULL should be counted.

Return Type:

INTEGER

Remarks:

- Aggregate functions are allowed as <u>field-expressions</u> only in the <u>SELECT</u>, <u>HAVING</u>, and <u>ORDER BY</u> clauses.
- The arguments of an aggregate function can not reference other aggregate functions.
- The arguments of an aggregate function can not reference the following <u>functions</u>:
 - <u>SEQUENCE</u>
 - <u>OUT_ROW_NUMBER</u>
- DISTINCT is allowed in aggregate functions only when there is no GROUP BY clause.

Examples:

A. COUNT(*)

The following query returns the total number of events in the System event log:

```
SELECT COUNT(*)
```

FROM System

B. COUNT(DISTINCT)

The following query returns the total number of distinct event source names in the System event log:

```
SELECT COUNT(DISTINCT SourceName)
```

FROM System

C. COUNT(*) and GROUP BY

The following query returns the total number of events generated by

each event source in the System event log:

SELECT SourceName, COUNT(*)

FROM System

DG ROUINB (/ field:@xpression)

The following query returns the total number of non-null values for the "cs-username" field in the specified <u>IIS W3C</u> log file:

SELECT COUNT(cs-username)

FROM ex040528.log

E. COUNT(*) and WHERE

The following query returns the total number of requests to a page logged in the specified $\underline{IIS W3C}$ log file:

SELECT COUNT(*)

FROM ex040528.log

FVCORNT(*),ICROUP'/BYDandpHAVING

The following query returns the pages in the specified <u>IIS W3C</u> log file that have been requested more than 50 times:

SELECT cs-uri-stem FROM ex040528.log GROUP BY cs-uri-stem HAVING COUNT(*) > 50 See also:

SUM AVG MAX MIN PROPCOUNT PROPSUM GROUPING

Aggregate Functions

Aggregating Data Within Groups

GROUPING

GROUPING (<field_expr>)

Returns a value of 1 when the row is added by the ROLLUP operator of the <u>GROUP BY</u> clause, or 0 when the row is not the result of ROLLUP. GROUPING is used to distinguish the NULL values returned by ROLLUP from standard NULL values. The NULL returned as the result of a ROLLUP operation is a special use of NULL. It acts as a value placeholder in the result set and means "all".

Arguments:

<field_expr>

The GROUP BY field-expression checked for null values.

Return Type:

INTEGER

Remarks:

- The GROUPING aggregate function is allowed only when the <u>GROUP</u> <u>BY</u> clause contains the ROLLUP operator.
- Aggregate functions are allowed as <u>field-expressions</u> only in the <u>SELECT</u>, <u>HAVING</u>, and <u>ORDER BY</u> clauses.
- The arguments of an aggregate function can not reference other aggregate functions.
- The arguments of an aggregate function can not reference the following <u>functions</u>:

- <u>SEQUENCE</u>
- OUT_ROW_NUMBER

Examples:

A. GROUPING

The following query, on an <u>IISW3C</u> log file, returns the number of requests for each page on each day, and uses the ROLLUP operator to also display summary rows showing the number of requests for each day, and the total number of requests:

SELECT date, cs-uri-stem, COUNT(*), GROUPING(date) AS GDate, G ROUPING(cs-uri-stem) AS GPage Arstannalexoutputsyloguld be: GROUP BY date, cs-uri-stem WITH ROLLUP COUNT(ALL *) GDate GPage cs-uri-stem date The yalves of the lighter field are honly for the rows in which the "date: fieldsiss Welds due to the introduction of the ROLLUP summary r**2005**3-11-18 /images/address.gif 1 0 0 Similarly, 189 calibre of the except age" field are 1 only for the rows in whigh-the1's shuit steps " field is NULL due to the introduction of the RODBUP BUNG Mary 169MSo.nsf 6 0 0 2003-11-20 /data/rulesinfo.nsf 5 0 0 2003-11-20 /maindefault.htm 1 0 0 2003-11-20 /top2.htm 0 1 0 See also: /homelog.swf 1 0 0 1 20 1 COUNT -0 SU2003-11-18 -6 1 6 AV2003-11-19 -0 1 MA2003-11-20 -8 0 1 MIN PROPCOUNT PROPSUM

GROUP BY Aggregate Functions

Aggregating Data Within Groups

MAX

MAX ([DISTINCT | ALL] <field_expr>)

Returns the maximum value among all the values of the specified <u>field-expression</u>.

Arguments:

DISTINCT

Specifies that MAX returns the maximum value of unique values. DISTINCT is not meaningful with MAX and is available for SQL-92 compatibility only.

DISTINCT can only be used when the query does not make use of the <u>GROUP BY</u> clause.

ALL

Applies the aggregate function to all values. ALL is the default.

<field_expr>

The <u>field-expression</u> among whose values the maximum is to be found.

The field-expression can be of any data type.

Return Type:

The returned type is the same as the argument field-expression.

Remarks:

- NULL values are ignored by the MAX aggregate function.
- Aggregate functions are allowed as <u>field-expressions</u> only in the <u>SELECT</u>, <u>HAVING</u>, and <u>ORDER BY</u> clauses.
- The arguments of an aggregate function can not reference other aggregate functions.
- The arguments of an aggregate function can not reference the following <u>functions</u>:
 - <u>SEQUENCE</u>
 - <u>OUT_ROW_NUMBER</u>
- DISTINCT is allowed in aggregate functions only when there is no GROUP BY clause.

Examples:

A. MAX

The following query returns the size of the largest executable file in the "system32" directory, using the <u>FS</u> input format:

SELECT MAX(Size)

FROM C:\windows\system32*.*

BWMAKEATO_GROUR BASE(EXTRACT_EXTENSION(Name)) = 'exe'

The following query returns the longest time spent by each page extension logged in the specified <u>IIS W3C</u> log file:

SELECT_TO_LOWERCASE(EXTRACT_EXTENSION(cs-uri-stem)) A

S PageType, MAX(time-taken) FROM ex031118.log Sec also: P BY PageType

COUNT SUM AVG

MIN PROPCOUNT PROPSUM GROUPING

Aggregate Functions

Aggregating Data Within Groups

MIN

MIN ([DISTINCT | ALL] <field_expr>)

Returns the minimum value among all the values of the specified <u>field-</u><u>expression</u>.

Arguments:

DISTINCT

Specifies that MIN returns the minimum value of unique values. DISTINCT is not meaningful with MIN and is available for SQL-92 compatibility only.

DISTINCT can only be used when the query does not make use of the <u>GROUP BY</u> clause.

ALL

Applies the aggregate function to all values. ALL is the default.

<field_expr>

The <u>field-expression</u> among whose values the minimum is to be found.

The field-expression can be of any data type.

Return Type:

The returned type is the same as the argument field-expression.

Remarks:

- NULL values are ignored by the MIN aggregate function.
- Aggregate functions are allowed as <u>field-expressions</u> only in the <u>SELECT</u>, <u>HAVING</u>, and <u>ORDER BY</u> clauses.
- The arguments of an aggregate function can not reference other aggregate functions.
- The arguments of an aggregate function can not reference the following <u>functions</u>:
 - <u>SEQUENCE</u>
 - <u>OUT_ROW_NUMBER</u>
- DISTINCT is allowed in aggregate functions only when there is no GROUP BY clause.

Examples:

A. MIN

The following query returns the size of the smallest executable file in the "system32" directory, using the FS input format:

```
SELECT MIN(Size)
```

FROM C:\windows\system32*.*

BWMMIREARCOGROMPREASE(EXTRACT_EXTENSION(Name)) = 'exe'

The following query returns the shortest and the longest time spent by each page extension logged in the specified <u>IIS W3C</u> log file:

```
SELECT TO_LOWERCASE(EXTRACT_EXTENSION(cs-uri-stem)) A
S PageType,
MIN(time-taken),
MAX(time-taken)
See RISM ex031118.log
COGROUP BY PageType
SUM
AVG
```

MAX PROPCOUNT PROPSUM GROUPING

Aggregate Functions

Aggregating Data Within Groups

PROPCOUNT

PROPCOUNT (*) [ON (<on_field_expr_list>)]
PROPCOUNT (<field_expr_list>) [ON (<on_field_expr_list>)]
<field_expr_list> ::= <field_expr> [, <field_expr> ...]
<on_field_expr_list> ::= <field_expr> [, <field_expr> ...]

Returns the ratio of the <u>COUNT</u> aggregate function calculated on a group to the COUNT aggregate function calculated on a hierarchically higher group.

Arguments:

*

Specifies that all records should be counted to return the total number of records, including records that contain NULL values.

<field_expr_list>

Specifies that only records for which at least one of the specified <u>field-expressions</u> is non-NULL should be counted.

<on_field_expr_list>

List of GROUP BY field-expressions identifying the hierarchically higher group on which the denominator COUNT aggregate function is to be calculated.

This list of field-expressions must be a *proper prefix* of the GROUP BY field-expressions, that is, it must contain, in the same order, a subset of the field-expressions specified in the GROUP BY clause, starting with the leftmost GROUP BY field-expression. When this list of field-expressions is not specified, the denominator COUNT aggregate function is calculated on the whole set of input records.

Return Type:

REAL

Remarks:

- When used without a <u>GROUP BY</u> clause, the PROPCOUNT aggregate function always returns 1.0. In fact, in this case the only hierarchically higher group available is the whole set of input records, and the ratio numerator and denominator are calculated on the same set.
- To obtain a percentage, multiply the return value of the PROPCOUNT aggregate function by 100.0, using the <u>MUL</u> function.
- Aggregate functions are allowed as <u>field-expressions</u> only in the <u>SELECT</u>, <u>HAVING</u>, and <u>ORDER BY</u> clauses.
- The arguments of an aggregate function can not reference other aggregate functions.
- The arguments of an aggregate function can not reference the following <u>functions</u>:
 - <u>SEQUENCE</u>
 - <u>OUT_ROW_NUMBER</u>

Examples:

A. PROPCOUNT(*)

The following query returns the percentage of events for each source in the System event log:

SELECT SourceName, MUL(PROPCOUNT(*), 100.0) AS Percent FROM System Acsample By tout of this guery is:

SourceName Percent

The "Recent" output record field shows the ratio of the number of eyentedoggebby/anger cestoothey total number of events in the event

QSti HotKey Poller 3.430691

Application Popup 0.108175

Inwhispexample, the calculation performed by the PROPCOUNT apgregate function is reggivalent to executing the following two questies and ealculating \$1944 atto of the two aggregate functions for each sevent log source 4525

RemoteAccess 2.194406

SEIMERCICSourceNam0, 509968T(*) AS Numerator

SROWS Stystem 0.509968

SECULT BY Spy 10:26434 Denominator

0.015454 FROMTSystem

0.030907 **B**^P^I**U**sing ON

0.077268 Ollowing query uses the <u>IISW3C Input Format</u> to parse IIS log files and calculate the percentage of hits for a page type and HTTP status code relative to the number of hits for that page type (i.e. the distribution of HTTP status codes within each page type):

SELECT EXTRACT_EXTENSION(cs-uri-stem) AS PageType, sc-status, MUL(PROPCOUNT(*) ON (PageType), 100.0) AS Hits AFstempelex04tps28gfbthis query is:

GROUP BY PageType, sc-status

PERPEKPERSY-BEERE Typies sc-status

For peach page type another TP status code, the "Hits" output record field shows the ratio of the number of requests for that page type and HeTase status code too the total number of requests for that page type.

13.636364 CSS 200

In this examples the adjustion performed by the PROPCOUNT

aggregate function 000000 quivalent to executing the following two quieries 5000 calouration of the two aggregate functions for each page type 10000000 status:

gif 200 21.025641

SIELE COP4EXT 花名 22 BOE XTENSION(cs-uri-stem) AS Page Type, sc-status, SECUN 494*) AS Page Type, SC-status, S

BEACT AND AND A STREET STENSION (cs-uri-stem) AS PageType, COUNT

PROMIE AND BERNELS

etterburgerype

See	₽₽₽₽₽	
ipg	304	77.922078
COUNT	200	36.363636
<u>SUM</u>	304	63.636364
AVG	200	90.845070
MAX fisf	302	0.704225
MIN	304	6.338028
PROPSI	<u>JM</u> 3	2.112676

GROUPING SWI 200 27.272727

Aggregate Functions 72,727273

Aggregating Data Within Groups Calculating Percentages

PROPSUM

PROPSUM (<field_expr>) [ON (<on_field_expr_list>)]

<on_field_expr_list> ::= <field_expr> [, <field_expr> ...]

Returns the ratio of the <u>SUM</u> aggregate function calculated on a group to the SUM aggregate function calculated on a hierarchically higher group.

Arguments:

<field_expr>

The <u>field-expression</u> whose values are to be summed. The field-expression data type must be <u>INTEGER</u> or <u>REAL</u>.

<on_field_expr_list>

List of GROUP BY field-expressions identifying the hierarchically higher group on which the denominator SUM aggregate function is to be calculated.

This list of field-expressions must be a *proper prefix* of the GROUP BY field-expressions, that is, it must contain, in the same order, a subset of the field-expressions specified in the GROUP BY clause, starting with the leftmost GROUP BY field-expression. When this list of field-expressions is not specified, the denominator SUM aggregate function is calculated on the whole set of input records.

Return Type:

REAL

Remarks:

- When used without a <u>GROUP BY</u> clause, the PROPSUM aggregate function always returns 1.0. In fact, in this case the only hierarchically higher group available is the whole set of input records, and the ratio numerator and denominator are calculated on the same set.
- To obtain a percentage, multiply the return value of the PROPSUM aggregate function by 100.0, using the <u>MUL</u> function.
- Aggregate functions are allowed as <u>field-expressions</u> only in the <u>SELECT</u>, <u>HAVING</u>, and <u>ORDER BY</u> clauses.
- The arguments of an aggregate function can not reference other aggregate functions.
- The arguments of an aggregate function can not reference the following <u>functions</u>:
 - <u>SEQUENCE</u>
 - <u>OUT_ROW_NUMBER</u>

Examples:

A. PROPSUM

The following query uses the <u>IISW3C Input Format</u> to parse IIS log files and calculate the percentage of bytes sent for each page type:

GROUP BY PageType

PageType PercentBytes

Then"Percent Bytes" output record field shows the ratio of the bytes sent for 1995 page type to the total number of bytes sent in the log. gif 23.772064 Inextais examples the calculation performed by the PROPSUM any fregate 450 mg ion is equivalent to executing the following two quefies 320 for 286 for a lating the ratio of the two aggregate functions for eargh page 06 / pto

html 0.104051

SELECTEXTENSION(cs-uri-stem) AS PageType, SUM(sc-Bynes) AS MOMERator

BLACTORIACS Detes) AS Denominator

FROM & DATE Type

B. Using ON

The following query uses the <u>IISW3C Input Format</u> to parse IIS log files and calculate the percentage of bytes sent for each page type and HTTP status code relative to the total bytes sent for that page type (i.e. the distribution of HTTP status code response bytes within each page type):

SELECT EXTRACT_EXTENSION(cs-uri-stem) AS PageType, sc-status, MUL(PROPSUM(sc-bytes) ON (PageType), 100.0) AS PercentBytes Arseonalex04tps28q6this query is:

GROUP BY PageType, sc-status

PBBEBBBC-BagesTypercentBytes

Fairpeachopage (type) and HTTP status code, the "PercentBytes" output record fights the ratio of the response bytes for that pages type and H.4083 status code to the total response bytes for that page type) 6.039609

css 304 3.502318

In this example of the scale gulation performed by the PROPSUM aggregate function is equivalent to executing the following two queries and cale dation to end of the two aggregate functions for each page type and 658TP status:

gif 304 6.935887

§HLE**404**EXT**R.2521**4<u>6</u>XTENSION(cs-uri-stem) AS PageType, sc-status, l**6h**dM(s**200**ytes)**92**(**526606**rator

5回口心中的水的花科107痘気TENSION(cs-uri-stem) AS PageType, SUM(sc-研究リ科学校会会研究の39c-status

CENTROPYCHROEMINT PERMONENTATION HAVING

Tery Uses the IISW3C Input Format to parse IIS log ge types that represent more than 10% of the sent: 2.036087 total byt 99.604883

 KTENSION(cs-uri-stem)
 AS
 PageType

 $UM(s_{6}-bytes) > 0.1$ See also: 304 0.089812 COUNT **SUM** AVG MAX MIN PROPCOUNT **GROUPING**

Aggregate Functions

Aggregating Data Within Groups Calculating Percentages

SUM

SUM ([DISTINCT | ALL] <field_expr>)

Returns the sum of all the values, or only the DISTINCT values, of the specified <u>field-expression</u>.

Arguments:

DISTINCT

Specifies that SUM returns the sum of unique values. DISTINCT can only be used when the query does not make use of the <u>GROUP BY</u> clause.

ALL

Applies the aggregate function to all values. ALL is the default.

<field_expr>

The <u>field-expression</u> whose values are to be summed. The field-expression data type must be <u>INTEGER</u> or <u>REAL</u>.

Return Type:

INTEGER or **REAL**, depending on the argument field-expression.

Remarks:

- NULL values are ignored by the SUM aggregate function.
- Aggregate functions are allowed as <u>field-expressions</u> only in the <u>SELECT</u>, <u>HAVING</u>, and <u>ORDER BY</u> clauses.

- The arguments of an aggregate function can not reference other aggregate functions.
- The arguments of an aggregate function can not reference the following <u>functions</u>:
 - <u>SEQUENCE</u>
 - OUT_ROW_NUMBER
- DISTINCT is allowed in aggregate functions only when there is no GROUP BY clause.

Examples:

A. SUM

The following query returns the total number of bytes for executable files in the "system32" directory, using the <u>FS</u> input format:

SELECT SUM(Size)

FROM C:\windows\system32*.*

BWSUMEATO_GROUP BYSE(EXTRACT_EXTENSION(Name)) = 'exe'

The following query returns the total number of bytes sent for each page extension logged in the specified <u>IIS W3C</u> log file:

SELECT TO_LOWERCASE(EXTRACT_EXTENSION(cs-uri-stem)) A S PageType, SUM(sc-bytes) FROM ex031118.log Sec also JP BY PageType

COUNT AVG MAX MIN PROPCOUNT PROPSUM GROUPING Aggregate Functions

Aggregating Data Within Groups

Functions

<function></function>	::=	<function_name> (<argument_list>)</argument_list></function_name>
<argument_list></argument_list>	::=	<field_expr> [, <field_expr>]</field_expr></field_expr>
		<empty></empty>

Log Parser functions take zero or more <u>field-expressions</u> as arguments, process the arguments, and return a single value.

Remarks:

 Generally, functions that take no arguments and functions whose arguments are <u>constant values</u> are executed and replaced with the return value before the query is processed.
 As an example, the following query uses a function with no arguments and a function with constant arguments:

SELECT COMPUTER_NAME(), SUM(4, 5), TimeGenerated

FROM System

Before being processed, the query is modified as follows:

SELECT 'MYSERVER0', 9, TimeGenerated

FROM System

The only zero-argument functions that are not replaced with their return value before the query is processed are:

- <u>SEQUENCE</u>
- <u>IN_ROW_NUMBER</u>
- <u>OUT_ROW_NUMBER</u>

Functions:

Arithmetical

ADD BIT AND BIT NOT BIT OR BIT SHL BIT_SHR BIT XOR DIV EXP EXP10 FLOOR LOG LOG10 MOD MUL QNTFLOOR_TO_DIGIT **ONTROUND TO DIGIT** QUANTIZE ROUND <u>SQR</u> **SORROOT** SUB

Conversion

HEX_TO_INT INT_TO_IPV4 IPV4_TO_INT TO_DATE TO_HEX TO_INT TO_LOCALTIME TO_REAL TO_STRING TO_TIME TO_TIMESTAMP TO_UTCTIME

String Manipulation

EXTRACT EXTENSION EXTRACT FILENAME EXTRACT PATH EXTRACT PREFIX EXTRACT SUFFIX EXTRACT TOKEN EXTRACT VALUE HEX_TO_ASC HEX TO HEX16 HEX TO HEX32 HEX TO HEX8 HEX TO PRINT INDEX OF LAST INDEX OF **LTRIM** REPLACE CHR REPLACE STR ROT13 RTRIM STRCAT **STRCNT** STRLEN **STRREPEAT STRREV** SUBSTR TO LOWERCASE TO UPPERCASE TRIM

URLESCAPE URLUNESCAPE

System Information

COMPUTER_NAME RESOLVE_SID REVERSEDNS SYSTEM_DATE SYSTEM_TIME SYSTEM_TIMESTAMP SYSTEM_UTCOFFSET

Miscellaneous

CASE COALESCE HASHMD5_FILE HASHSEQ IN_ROW_NUMBER OUT_ROW_NUMBER REPLACE_IF_NOT_NULL SEQUENCE WIN32_ERROR_DESCRIPTION

Note: The **REPLACE_IF_NULL** function has been deprecated in favor of the <u>COALESCE</u> function.

See also:

Aggregate Functions

Constant Values

<value></value>	::=	<integer_constant></integer_constant>
		<real_constant></real_constant>
		<string_constant></string_constant>
		<timestamp_constant></timestamp_constant>
		<null_constant></null_constant>
<integer_constant></integer_constant>	::=	integer 0xhexadecimal
<real_constant></real_constant>	::=	integer_part.fractional_part
<string_constant></string_constant>	::=	'string'
<timestamp_constant></timestamp_constant>	::=	TIMESTAMP ('timestamp' , 'format')
<null_constant></null_constant>	::=	NULL

Constants are immutable <u>field-expressions</u>, and they are mostly used in <u>expressions</u> or as arguments of <u>functions</u>.

Constants:

<integer_constant>

Constant values of the INTEGER type can be entered as decimal numbers, or as hexadecimal numbers preceded by the "0x" prefix. For more information about the Log Parser INTEGER data type, see INTEGER Data Type.

<real_constant>

Constant values of the REAL type are entered as decimal numbers containing a decimal point.

For more information about the Log Parser REAL data type, see <u>REAL Data Type</u>.

<string_constant>

Constant values of the STRING type are entered as strings enclosed by single quote characters (').

The single quote character (') and the backslash character ($\$) are considered special characters in a string constant, and they can only be entered as escape sequences preceded by a backslash character ($\$ and $\$), as in the following example:

'Contains \' single quote and \\ backslash'

In addition, any character (including illegal characters and nonprintable characters) can be entered using the **\uxxxx** notation, where *xxxx* is the 4-digit hexadecimal representation of the desired UNICODE character, as in the following example:

'Contains\u0009tabs'

For more information about the Log Parser STRING data type, see <u>STRING Data Type</u>.

<timestamp_constant>

Constant values of the TIMESTAMP type are entered with the special **TIMESTAMP** keyword, followed by a string representation of the desired timestamp, and by the format of the string representation of the desired timestamp, using the Log Parser <u>Timestamp Format</u> <u>Specifiers</u>.

If the timestamp format specifiers include date specifiers only, the resulting TIMESTAMP value will be a date-only timestamp. Similarly, if the timestamp format specifiers include time of day specifiers only, the resulting TIMESTAMP value will be a time-only timestamp. For more information about the Log Parser TIMESTAMP data type,
see <u>TIMESTAMP Data Type</u>.

<null_constant>

Constant values of the NULL type are entered with the special **NULL** keyword.

For more information about the Log Parser NULL data type, see <u>NULL Data Type</u>.

Remarks:

 Integer constants entered as hexadecimal numbers are converted internally to decimal values. To force an output format to display an integer field-expression as an hexadecimal value, use the <u>TO_HEX</u> function.

Examples:

A. Integer constant entered as decimal number

sc-bytes >= 1000

B. Integer constant entered as hexadecimal number

BIT_AND(Flags, 0x1000)

C. Real constant

AVG(time-taken) < 75.45

D. String constant

'Some string'

E. String constant containing special characters

'Contains \' single quote and \\ backslash'

F. String constant containing UNICODE characters

'Contains a \u2530 UNICODE character'

G. Timestamp constant

TimeGenerated > TIMESTAMP('2004-05-28 19:12:43', 'yyyy-MM-dd h h:mm:ss')

H. Date-only timestamp constant

date > TIMESTAMP('2004-05-28', 'yyyy-MM-dd')

I. Time-only timestamp constant

time > TIMESTAMP('19:12:43', 'hh:mm:ss')

J. NULL constant

Message <> NULL

See also:

Field Expressions INTEGER Data Type REAL Data Type STRING Data Type TIMESTAMP Data Type NULL Data Type

Basics of a Query

Comments

::=

<comment>

/* text_of_comment */
-- text_of_comment

Comments are user-provided text not evaluated by Log Parser, used to document code or temporarily disable parts of query statements.

Remarks:

- Use -- for single-line or nested comments. Comments inserted with -- are delimited by the newline character.
- Multiple-line comments must be indicated by /* and */.
- There is no maximum length for comments.

Examples:

A. Single-line comments

SELECT TimeGenerated, SourceName

FROM System -- We are using the SYSTEM event log

B. Multiple-line comments

SELECT TypeName, COUNT(*) AS TotalCount

USING TO_UPPERCASE(EXTRACT_TOKEN(EventTypeName, 0, '

')) AS TypeName © 2004 Microsoft Corporation. All rights reserved. INTO Report.csv

FROM System

/* We only want to retrieve event logs whose

type name contains 'service'

*/

WHERE TypeName LIKE '%service%'GROUP BY TypeNameHAVING TotalCount > 5ORDER BY TotalCount DESC

Data Types

In the Log Parser SQL-Like language, each <u>field-expression</u> has a related data type, which is an attribute that specifies the type of data that the field-expression can hold.

Log Parser supplies a set of system data types that define all of the types of data that can be used with Log Parser. The set of system-supplied data types is:

- **INTEGER**: integer numeric data;
- **<u>REAL</u>**: floating precision numeric data;
- <u>STRING</u>: variable length UNICODE character string data;
- **<u>TIMESTAMP</u>**: date and time data;
- <u>NULL</u>: unknown or unavailable data.

INTEGER Data Type

The **INTEGER** data type represents integer (whole number) numeric data.

Value range:

INTEGER values are represented as signed 64-bit (8-byte) integer numbers, with values ranging from -2^63 (-9,223,372,036,854,775,808) through 2^63-1 (9,223,372,036,854,775,807).

Conversion Functions:

Other data types to INTEGER data type:

• <u>TO_INT</u>

INTEGER data type to other data types:

- <u>TO_REAL</u>
- TO_STRING
- <u>TO_TIMESTAMP</u>

See also:

Constant Values

REAL Data Type

The **REAL** data type represents floating point numeric data. Floating point data is approximate; not all values in the data type range can be precisely represented.

Value range:

REAL values are represented as signed 64-bit (8-byte) floating point numbers, with values ranging from $\pm 5.0 \times 10^{-324}$ through $\pm 1.7 \times 10^{308}$, with at least 15 digits of precision.

Conversion Functions:

Other data types to REAL data type:

• <u>TO_REAL</u>

REAL data type to other data types:

- <u>TO_INT</u>
- <u>TO_STRING</u>
- <u>TO_TIMESTAMP</u>

See also:

Constant Values

STRING Data Type

The **STRING** data type represents variable length UNICODE character string data.

Conversion Functions:

Other data types to STRING data type:

• <u>TO_STRING</u>

STRING data type to other data types:

- <u>TO_INT</u>
- <u>TO_REAL</u>
- <u>TO_TIMESTAMP</u>

See also:

Constant Values

TIMESTAMP Data Type

The **TIMESTAMP** data type represents date and time of day data.

Value range:

TIMESTAMP values range from January 1, -8192 through December 31, 8191, to an accuracy of one hundred nanoseconds (one ten-thousandth of a millisecond).

Date-only and Time-only Timestamps

TIMESTAMP values can be restricted to represent date data only or time of day data only.

As explained in the Remarks section below, a TIMESTAMP value that has been restricted to represent date data only or time of day data only will be formatted to display date elements only (year, month, and day) or time of day elements only (hour, minute, second, millisecond, and nanosecond).

TIMESTAMP values can be restricted to date-only or time-only timestamps in different ways.

Some input formats return TIMESTAMP input record fields whose values represent only dates or times of day. For example, the "date" and "time" fields of the <u>IISW3C</u> input format have values representing only dates and times of day, respectively.

TIMESTAMP <u>constants</u> can also be entered as date-only or timeonly timestamp values, depending on the <u>Timestamp Format</u> <u>Specifiers</u> used.

In addition, the <u>TO_DATE</u>, <u>TO_TIME</u>, <u>SYSTEM_DATE</u>, and <u>SYSTEM_TIME</u> functions all return TIMESTAMP values representing dates or times of day only.

For more information, refer to the Remarks section below.

Remarks:

- TIMESTAMP values are formatted and parsed using <u>Timestamp</u> <u>Format Specifiers</u>. Timestamp format specifiers are strings that use special characters to describe date and/or time elements in a string representation of a timestamp. For more information, refer to the <u>Timestamp Format Specifiers</u> reference.
- Although the distinction between date-only or time-only TIMESTAMP values and full TIMESTAMP values is often transparent to the user, date-only or time-only values behave differently than full TIMESTAMP values in the following circumstances:
 - Comparison operators in <u>expressions</u>: When comparing a date-only TIMESTAMP value with another TIMESTAMP value, the time of day data of the date-only value is assumed to be time zero. Similarly, when comparing a time-only TIMESTAMP value with another TIMESTAMP value, the date data of the time-only value is assumed to be January 1, year 0.
 - Formatting TIMESTAMP values: whenever a date-only or time-only TIMESTAMP value is formatted to a STRING value by either explicitly using the <u>TO_STRING</u> function or as implicitly done by an output format, the resulting STRING will only contain the date or time of day data, and the non-applicable <u>Timestamp Format Specifiers</u> will be ignored.

As an example, the following query uses the TO_STRING function with date and time of day format specifiers to format the "time" field of the IISW3C input format:

SELECT TO_STRING(time, 'yyyy-MM-dd hh:mm:ss')

FROM <1>

Since the values of the "time" field are time-only TIMESTAMP values, the resulting STRING values will be formatted according to the time of day format specifiers only, and the date format specifiers will be ignored:

```
18:48:04
```

18:48:27

• Values40f2type TIMESTAMP can also be used to represent time intervals; for example with the ADD and SUB functions.

Since the origin of time in the Log Parser SQL-Like language is

January 1, year 0, time intervals should be expressed as timestamps relative to this origin of time.

For example, a time interval of one day should be specified as January 2, year 0, i.e. 24 hours after the origin of time.

The following example query selects all the event log records that have been written in the past 2 days:

SELECT *

FROM SYSTEM

 TWHESTAME with Ends not carcy information (ious (stysticnazonates ramp timestames is Note (bootpol-03', 'yyyy-MM-dd')))
 When working with TIMESTAMP fields generated by an input format, users should be aware of the timezone these fields are relative to, and handle their values accordingly.
 For example, values of the "TimeGenerated" field of the EVT Input Format are relative to the local timezone. If Universal Time Coordinates (UTC) are desired, the TO_UTCTIME function should be used to

convert these local timestamps to UTC timestamps.

Conversion Functions:

Other data types to TIMESTAMP data type:

• <u>TO_TIMESTAMP</u>

TIMESTAMP data type to other data types:

- <u>TO_INT</u>
- <u>TO_REAL</u>
- <u>TO_STRING</u>

Full TIMESTAMP values to date-only TIMESTAMP values:

• <u>TO_DATE</u>

Full TIMESTAMP values to time-only TIMESTAMP values:

• <u>TO_TIME</u>

Date-only and time-only TIMESTAMP values to full TIMESTAMP values:

• <u>TO_TIMESTAMP</u>

Local timezone TIMESTAMP values to UTC TIMESTAMP values:

• <u>TO_UTCTIME</u>

UTC TIMESTAMP values to local timezone TIMESTAMP values:

• <u>TO_LOCALTIME</u>

See also:

<u>Constant Values</u> <u>Timestamp Format Specifiers</u>

Timestamp Format Specifiers

TIMESTAMP values are formatted and parsed using **Timestamp Format Specifiers**. Timestamp format specifiers are strings that use special characters to describe date and/or time elements in a string representation of a timestamp.

Timestamp format specifiers are used in the following circumstances:

• When entering a TIMESTAMP <u>constant</u> with the TIMESTAMP keyword. In this case, timestamp format specifiers are used to describe how the string entered should be parsed in order to obtain a TIMESTAMP value, as in the following example:

TimeGenerated > TIMESTAMP ('2004-05-28 10:23:15', 'yyyy-MM-dd hh: mm:ss')

• When converting a TIMESTAMP value to a STRING value using the <u>TO_STRING</u> function. In this case, timestamp format specifiers are used to describe how the TIMESTAMP value should be formatted in order to obtain a STRING value, as in the following example:

TO_STRING(TimeGenerated, 'yyyy MMM, dd h:m:s')

• When converting a STRING value to a TIMESTAMP value using the <u>TO_TIMESTAMP</u> function. In this case, timestamp format specifiers are used to describe how the STRING value should be parsed in order to obtain a TIMESTAMP value, as in the following example:

TO_TIMESTAMP(Text, 'MMM ddd yyyy')

• When specifying how an input format should parse TIMESTAMP fields, using the "iTsFormat" parameter. In this case, timestamp format specifiers are used to describe how timestamp values are represented by the selected data source, so that the input format is capable to parse these fields and represent them as values of type TIMESTAMP. The following example sets a specific value for the "iTsFormat"

parameter of the <u>CSV Input Format</u>:

C:\>logparser "SELECT MyField FROM file.csv" -i:CSV -iTsFormat:"yyyy -MM-dd"

 When specifying how an output format should format and display TIMESTAMP fields, using the "oTsFormat" parameter. In this case, timestamp format specifiers are used to describe how TIMESTAMP values should be formatted by the output format, as in the following example using the <u>TSV Output Format</u>:

C:\>logparser "SELECT TimeGenerated INTO file.txt FROM System" -i:E VT -o:TSV -oTsFormat:"yyyy-MM-dd"

The following table describes the timestamp format specifiers supported by the Log Parser SQL-Like language:

Specifier	Description	Example specifier strings	Example formats
У	year, last digit (when parsing, assumed to be relative to year 2000)	y MM dd	4 05 28
УУ	year, last 2 digits (when parsing, assumed to be relative to year 2000)	yy MM dd	04 05 28
ууу	year, last 3 digits (when parsing, assumed to be relative to year 2000)	yyy MM dd	004 05 28
уууу	year, 4 digits	yyyy MM dd	2004 05 28
Μ	month, no leading zero	yyyy-M-dd	2004-5-28 2004-12-01
MM	month, leading zero	yyyy-MM-dd	2004-05-28 2004-12-01

MP	month, leading space	yyyy-MP-dd	2004- 5-28 2004-12-01
МХ	month, with or without leading zero (when parsing) month, without leading zero (when formatting)		2004-05-28 (when parsing) 2004-5-28 2004-12-01
МММ	month, 3-character abbreviation of name (1)	MMM d, yyyy	Dec 1, 2004
ММММ	month, full name (1)	MMMM d, уууу	December 1, 2004
d	day, no leading zero	yyyy-MM-d	2004-12-1 2004-05-28
dd	day, leading zero	yyyy-MM-dd	2004-12-01 2004-05-28
dp	day, leading space	уууу-ММ-dp	2004-12- 1 2004-05-28
dx	day, with or without leading zero (when parsing) day, without leading zero (when formatting)	yyyy-MM-dx	2004-12-01 (when parsing) 2004-12-1 2004-05-28
ddd	week day, 3-character abbreviation of name (1)	ddd MMMM d, УУУУ	Wed December 1, 2004
dddd	week day, full name (1)	dddd MMMM d, yyyy	Wednesday December 1, 2004
h, H	hour, no leading zero	h:mm:ss	3:12:05 21:04:15
hh, HH	hour, leading zero	hh:mm:ss	03:12:05 21:04:15
hp, HP	hour, leading space	hp:mm:ss	3:12:05 21:04:15

hx, HX	hour, with or without leading zero (when parsing) hour, without leading zero (when formatting)	hx:mm:ss	03:12:05 (when parsing) 3:12:05 21:04:15
m	minute, no leading zero	hh:m:ss	21:4:15 03:12:05
mm	minute, leading zero	hh:mm:ss	21:04:15 03:12:05
mp	minute, leading space	hh:mp:ss	21: 4:15 03:12:05
mx	minute, with or without leading zero (when parsing) minute, without leading zero (when formatting)	hh:mx:ss	21:04:15 (when parsing) 21:4:15 3:12:05
S	second, no leading zero	hh:mm:ss	03:12:5 21:04:15
SS	second, leading zero	hh:mm:ss	03:12:05 21:04:15
sp	second, leading space	hh:mm:sp	03:12: 5 21:04:15
SX	second, with or without leading zero (when parsing) second, without leading zero (when formatting)	thh:mm:ss	03:12:05 (when parsing) 03:12:5 21:04:15
1	millisecond, no leading zeroes	hh:mm:ss.l	21:4:15.5 03:12:05.395
11	millisecond, leading zeroes	hh:mm:ss.ll	21:04:15.005 03:12:05.395
lp	millisecond, leading spaces	hh:mm:ss.lp	21:04:15. 5 03:12:05.395

lx		millisecond, with or without leading zero (when parsing) millisecond, without leading zero (when formatting)	hh:mm:ss.lx	21:04:15.005 (when parsing) 21:04:15.5 3:12:05.395
n		nanosecond, no leading zeroes	hh:mm:ss.ll.n	21:4:15.005.400 03:12:05.395.1900
nn		nanosecond, leading zeroes	hh:mm:ss.ll.nn	21:04:15.005.00000400 03:12:05.395.001900
np		nanosecond, leading spaces	hh:mm:ss.ll.np	21:04:15.005. 400 03:12:05.395. 1900
nx		nanosecond, with or without leading zero (when parsing) nanosecond, without leading zero (when formatting)	hh:mm:ss.ll.nx	21:04:15.005.00000400 (when parsing) 21:04:15.005.400 3:12:05.395.1900
tt		AM/PM notation	hh:mm:ss tt	09:04:15 PM 03:12.05 AM
?		any character (when parsing) space (when formatting)	yyyy-MM-dd? hh:mm:ss	2004-05-28T21:04:15 (when parsing) 2004-05-28 21:04:15 (when formatting)
any char	other racter	verbatim character	hh:mm:ss yyyy.MM+dd	09:04:15 2004.05+28

Notes:

(1): element names are obtained from the current system locale.

Date-only and Time-only Timestamps

When parsing a timestamp string, the following assumptions are made:

• If the timestamp format specifiers include date elements only, the resulting TIMESTAMP value will be a date-only timestamp; for example, the following statement creates a date-only TIMESTAMP constant value:

TIMESTAMP('2004-05-28', 'yyyy-MM-dd')

• If the timestamp format specifiers include time of day elements only, the resulting TIMESTAMP value will be a time-only timestamp; for example, the following statement creates a time-only TIMESTAMP constant value:

TIMESTAMP('21:04:15', 'hh:mm:ss')

• Unspecified date elements are replaced with the corresponding elements of the Log Parser origin date (January 1, year 0), unless the timestamp is a time-only timestamp value; for example, the following statement creates a date-only timestamp representing the date February 1, year 0:

TIMESTAMP('2', 'M')

Similarly, unspecified time elements are replaced with zero values, unless the timestamp is a date-only timestamp value; for example, the following statement creates a time-only timestamp representing the time 10:00:00.0.0:

TIMESTAMP('10', 'h')

As another example, the following statement creates a full timestamp value representing the time 10:00:00.0.0 on February 1, year 0:

TIMESTAMP('2 10', 'M h')

For more information on date-only and time-only timestamp values, refer to the <u>Timestamp Data Type</u> reference.

See also:

Constant Values Timestamp Data Type

NULL Data Type

The NULL data type represents unknown or unavailable data.

Remarks:

- Input formats often return NULL values for input record fields to indicate that the field data is not available in the current log.
- A value of NULL is different from a zero value. In the Log Parser SQL-Like language, comparison operators in <u>expressions</u> treat NULL values as the minimum possible values. In other words, all non-NULL values, even negative numeric values, are always greater than a NULL value. On the other hand, the <u>MIN</u> and <u>MAX</u> aggregate functions treat NULL values as respectively the maximum and minimum possible values. In other words, the MIN or MAX value between a non-NULL value and a NULL value is always the non-NULL value.
- To test for NULL values in a query use <u>IS NULL</u> or <u>IS NOT NULL</u> in the <u>WHERE</u> or <u>HAVING</u> clauses.

See also:

Constant Values Expressions

Input Formats

IIS Log File Input Formats

- <u>IISW3C</u>: parses IIS log files in the W3C Extended Log File Format.
- <u>IIS</u>: parses IIS log files in the Microsoft IIS Log File Format.
- **BIN**: parses IIS log files in the Centralized Binary Log File Format.
- <u>IISODBC</u>: returns database records from the tables logged to by IIS when configured to log in the ODBC Log Format.
- <u>HTTPERR</u>: parses HTTP error log files generated by Http.sys.
- <u>URLSCAN</u>: parses log files generated by the URLScan IIS filter.

Generic Text File Input Formats

- <u>CSV</u>: parses comma-separated values text files.
- <u>TSV</u>: parses tab-separated and space-separated values text files.
- XML: parses XML text files.
- <u>W3C</u>: parses text files in the W3C Extended Log File Format.
- <u>NCSA</u>: parses web server log files in the NCSA Common, Combined, and Extended Log File Formats.
- <u>TEXTLINE</u>: returns lines from generic text files.
- **<u>TEXTWORD</u>**: returns words from generic text files.

System Information Input Formats

- EVT: returns events from the Windows Event Log and from Event Log backup files (.evt files).
- <u>FS</u>: returns information on files and directories.
- <u>REG</u>: returns information on registry values.
- <u>ADS</u>: returns information on Active Directory objects.

Special-purpose Input Formats

- <u>NETMON</u>: parses network capture files created by NetMon.
- <u>ETW</u>: parses Enterprise Tracing for Windows trace log files and live sessions.
- <u>COM</u>: provides an interface to Custom Input Format COM Plugins.

ADS Input Format

The ADS input format returns properties of Active Directory objects.

The ADS input format enumerates the Active Directory objects in the Active Directory Container whose LDAP path is specified in the <u>from-</u><u>entity</u>, eventually recursing into additional Container objects found during the enumeration.

The information returned for each object depends on the value specified for the <u>objClass</u> parameter.

When the <u>objClass</u> parameter is left unspecified, the ADS input format works in "property mode", returning a record for each property of each object visited during the enumeration.

In this case, input records have a fixed number of fields whose values describe the properties being returned, including a "PropertyName" field and a "PropertyValue" field containing the name and the value of the property being processed.

Queries operating in "property mode" can work on Active Directory objects of different types, and since each input record represents a single object property, they can only reference a single property at a time.

For example, the following command returns the values of all the properties named "comment" from all the objects in the specified path:

LogParser "SELECT PropertyValue FROM LDAP://mydomain.mycompany.c om WHERE PropertyName = 'comment''' -i:ADS The output would look like the following example:

PropertyValue

Builtin

When the name of an Active Directory object class is specified for the <u>object ass</u> parameter, the ADS input format works in "object mode", returning a record for each object visited during the enumeration that is an instance of the specified class.

In this case, there is an input record field for each of the properties of the

oBjædtupeingnætursned.

Quality Operating in "object mode" can only work on Active Directory objects of a single type, and since each input record represents a single object, they can reference multiple properties of the same object at the same time.

For example, the following command returns the specified properties from all the objects of type "Computer":

LogParser "SELECT cn, operatingSystem, operatingSystemServicePack FRO M LDAP://mydomain.mycompany.com/CN=Computers,DC=mydomain,DC= The Contract of the contrac

cn	operatingSystem	operatingSystemServicePack
SERV	ER02 Windows XP	Professional Service Pack 1 Professional Service Pack 2
Fields Fields Param EXamo	MACHINE1 Windows MACHINE2 Windows MACHINE3 Windows	Server 2003 - S XP Professional Service Pack 2 S XP Professional Service Pack 1 S 2000 Server Service Pack 4

ADS Input Format From-Entity Syntax

<from- ::= [[<provider>:]//[<username>:<password>@] entity> <domain>]/<path>[;...]

The <u><from-entity></u> specified in queries using the ADS input format is a semicolon-separated list of LDAP paths.

Each LDAP path begins with an optional provider name (e.g. "IIS", "LDAP"), followed by an optional domain or computer name. If a provider name is not specified, then "IIS" is assumed by default. If a domain name or computer name is not specified, then "localhost" is assumed by default.

The from-entity can optionally include a username and a password to be used for the connection to the Active Directory provider. When these are not specified, the ADS input format uses the current user's credentials.

Note: LDAP paths containing comma (,) characters should be enclosed within single-quote (') characters.

Examples:

FROM IIS://COMPUTER01/W3SVC/1

FROM IIS://MyUsername:MyPassword@COMPUTER01/W3SVC/1

FROM 'LDAP://MyDomain/CN=Users,DC=MyDomain,DC=com'

FROM 'LDAP://MyUsername:MyPassword@MyDomain/CN=Users,DC=My Domain,DC=com'

FROM /W3SVC/1;/W3SVC/2;//COMPUTER02/W3SVC/1

ADS Input Format Fields

The structure of the input records generated by the ADS input format depends on the value specified for the <u>objClass</u> parameter.

Property Mode

When the <u>objClass</u> parameter is left unspecified, the ADS input format works in "property mode", returning a record for each property of each object visited during the enumeration.

In this case, input records have the following fixed structure:

Name	Туре	Description
ObjectPath	STRING	Full Active Directory path of the object containing this property
ObjectName	STRING	Name of the object containing this property
ObjectClass	STRING	Class name of the object containing this property
PropertyName	STRING	Name of the property being processed
PropertyValue	STRING	Value of the property being processed
PropertyType	STRING	Type of the property being processed

Queries operating in "property mode" can work on Active Directory objects of different types, and since each input record represents a single object property, they can only reference a single property at a time.

Object Mode

When the name of an Active Directory object class is specified for the <u>objClass</u> parameter, the ADS input format works in "object mode", returning a record for each object visited during the enumeration that is an instance of the specified class.

In this case, the first input record field is fixed, and it is described in the following table:

Name	Туре	Description
ObjectPath		Full Active Directory path of the object being processed

This field is followed by fields representing all the properties of the specified object class. Each field is named after the corresponding property name, and its data type is determined by the property type declared by the Active Directory schema object for the specified class.

Queries operating in "object mode" can only work on Active Directory objects of a single type, and since each input record represents a single object, they can reference multiple properties of the same object at the same time.

ADS Input Format Parameters

The ADS input format supports the following parameters:

objClass

Values:	Active Directory object class name
Default:	not specified
Description:	Object class name for "object mode" operation.
Details:	When this parameter is left unspecified, the ADS input fc "property mode", returning a record for each property of visited during the enumeration. On the other hand, when the name of an Active Directory specified for this parameter, the ADS input format works mode", returning a record for each object visited during t that is an instance of the specified class. For more information on the different modes of operation <u>Format Fields</u> .

Example: -objClass:User

username

Values:	username
values:	usemame

- Default: *not specified*
- Description: Username for the Active Directory connection.
- Details: When a username is not specified for this parameter, the format uses the username specified in the <u>from-entity</u> of from-entity does not include a username, the ADS input the current user's credentials.

Note: For security reasons, values specified for this parameter a not persisted when using the Log Parser command-line <u>Defaults</u> <u>Override Mode</u>.

	Example:	-username:MyUser
	Lixample.	-username.wryOser
pa	ssword	
	Values:	password
	Default:	not specified
	Description:	Password for the Active Directory connection.
	Details:	Password for the username specified with the "username
		Note : For security reasons, values specified for this parameter a not persisted when using the Log Parser command-line <u>Defaults</u> <u>Override Mode</u> .
	Example:	-password:MyPassword
rec	curse	
	Values:	recursion level (number)
	Default:	-1
	Description:	Max ADS container recursion level.
	Details:	0 disables container recursion; -1 enables unlimited recu
	Example:	-recurse:2

multiValuedSep

Values:	any string
Default:	I
Description:	Separator between values of multi-valued types.
Details:	Multi-valued property values are returned as a single stri concatenating the multiple values one after the other usi this parameter as a separator between the elements.
Example:	-multiValuedSep:,

ignoreDSErrors

Values:	ON OFF
Default:	ON
Description:	Ignore Directory Service errors.
Details:	When this parameter is set to "OFF", Directory Service e during the enumeration of objects and properties are retu <u>Errors</u> . When this parameter is set to "ON", Directory Service er ignored, and input record fields corresponding to unretrie properties are returned as NULL values.
Example:	-ignoreDSErrors:OFF

parseBinary

Values:	ON OFF
Default:	OFF
Description:	Return value of binary properties.
Details:	This parameter specifies whether properties containing k returned or not. When this parameter is set to "ON", binary values are re STRING values formatted according to the value specific "binaryFormat" parameter.
Example:	-parseBinary:ON

binaryFormat

Values:	ASC	PRINT	HEX
---------	-----	-------	-----

Default: HEX

Description: Format of binary properties.

Details: When the "parseBinary" property is set to "ON", the ADS returns properties containing binary values. In this case, are returned as STRING values formatted according to t specified for this parameter.
 When this parameter is set to "ASC", data bytes belongin 0x7F range are returned as ASCII characters, while data the range are returned as period (.) characters, as shown example:

Bucket: 02096553..rundll32.exe

When this parameter is set to "PRINT", data bytes repres ASCII characters are returned as ASCII characters, while do not represent printable ASCII characters are returned characters, as shown in the following example:

Bucket: 02096553

rundll32.exe

When this parameter is set to "HEX", all data bytes are r digit hexadecimal values, as shown in the following exan

4275636B65743A2030323039363535330D0A72756E646C6

Example:

-binaryFormat:PRINT

ADS Input Format Examples

Users' Job Titles

Retrieve users' job title breakdown from Active Directory:

LogParser "SELECT title, MUL(PROPCOUNT(*), 100.0) AS Percentage INT O DATAGRID FROM 'LDAP://MyUsername:MyPassword@mydomain/CN= Users,DC=mydomain,DC=com' WHERE title IS NOT NULL GROUP BY titl e ORDER BY Percentage DESC" -objClass:User **IIS AccessFlags MetaBase Properties**

Retrieve all the AccessFlags properties from IIS metabase objects:

LogParser "SELECT ObjectPath, PropertyValue FROM IIS://localhost WHER E PropertyName = 'AccessFlags'"

BIN Input Format

The BIN input format parses IIS log files in the Centralized Binary Log File Format.

When an IIS 6.0 web server is configured to log in the Centralized Binary Log File Format, all the IIS virtual sites hosted by the server log in a single, server-wide log file. Log files in this format are binary files, and the information contained in these logs can not be visualized by standard text file processors.

<u>From-Entity Syntax</u> <u>Fields</u> <u>Examples</u>

BIN Input Format From-Entity Syntax

<from-entity></from-entity>	::=	<filename> <siteid> [, <filename> <siteid>]</siteid></filename></siteid></filename>
<siteid></siteid>	::=	< site_number > < server_comment > < site_metabase_path >

The <from-entity> specified in queries using the BIN input format is a comma-separated list of:

- Paths of IIS Centralized Binary log files;
- IIS Virtual Site "identifiers".

"Site identifiers" must be enclosed within angle brackets (< and >), and can have one of the following values:

- The numeric site ID (e.g. "<1>", "<28163489>");
- The text value of the "ServerComment" property of the site (e.g. "<My External Site>", "<www.margiestravel.com>");
- The fully-qualified ADSI metabase path to the site (e.g. " <//MYSERVER/W3SVC/1>"), using either the numeric site ID or the text value of the "ServerComment" property of the site.

When a "site identifier" is used, the BIN input format connects to the specified machine's metabase, gathers information on the server's current logging properties, and parses all the log files in the server's current log file directory, returning only the entries corresponding to requests to the specified virtual site.

Filenames and "Site identifiers" can also include wildcards (e.g. "LogFiles\ra04*.ibl", "<www.*.com>").

Examples:

FROM LogFiles\ra04*.ibl, LogFiles\ra03*.ibl, \\MyServer\LoggingShare\W3S VC\ra04*.ibl

FROM <1>, <2>, <My External Site>, raw9.ibl

FROM <www.net*home.com>, <//MyServer2/W3SVC/www.net*home.com>, </*>
BIN Input Format Fields

The input records generated by the BIN input format contain the following fields:

Name	Туре	Description
LogFilename	STRING	Full path of the log file containing this entry
LogRow	INTEGER	Line in the log file containing this entry
ComputerName	STRING	The name of the server that served the request
SiteID	INTEGER	The IIS virtual site instance number that served the request
DateTime	TIMESTAMP	The date and time at which the request was served (Universal Time Coordinates (UTC) time)
ClientIpAddress	STRING	The IP address of the client that made the request
ServerIpAddress	STRING	The IP address of the server that served the request
ServerPort	INTEGER	The server port number that received the request
Method	STRING	The HTTP request verb

ProtocolVersion	STRING	The HTTP version of the client request
ProtocolStatus	INTEGER	The response HTTP status code
SubStatus	INTEGER	The response HTTP sub-status code
TimeTaken	INTEGER	The number of milliseconds elapsed since the moment the server received the request to the moment the server sent the last response chunk to the client
BytesSent	INTEGER	The number of bytes in the response sent by the server
BytesReceived	INTEGER	The number of bytes in the request sent by the client
Win32Status	INTEGER	The Windows status code associated with the response HTTP status code
UriStem	STRING	The HTTP request uri-stem
UriQuery	STRING	The HTTP request uri-query, or NULL if the requested URI did not include a uri-query
UserName	STRING	The name of the authenticated user that made the request, or NULL if the request was from an anonymous user

BIN Input Format Examples

Top 20 URL's for a Site

Create a chart containing the TOP 20 URL's in the "www.margiestravel.com" web site (assumed to be logging in the Centralized Binary log format):

LogParser "SELECT TOP 20 UriStem, COUNT(*) AS Hits INTO MyChart.gi f FROM <www.margiestravel.com> GROUP BY UriStem ORDER BY Hits D ESC" -chartType:Column3D -groupSize:1024x768

COM Input Format

The COM input format provides an interface to <u>Custom Input Format</u> <u>COM Plugins</u>.

With the Log Parser command-line executable, Custom Input Format COM Plugins are used through the COM input format. This input format takes the ProgID of the plugin COM object as a value of the <u>iProgID</u> parameter, and it provides an interface for command-line operations to use the custom input format.

With the Log Parser scriptable COM components, Custom Input Format COM Plugin objects can be used directly as arguments to the <u>Execute</u> or <u>ExecuteBatch</u> methods of the <u>LogQuery</u> object.

For this reason, the COM input format is not provided as a Log Parser scriptable COM component.

<u>From-Entity Syntax</u> <u>Fields</u> <u>Parameters</u> <u>Examples</u>

See also:

<u>Custom Plugins</u> <u>COM Input Format Plugins Reference</u>

COM Input Format From-Entity Syntax

The <<u>from-entity</u>> specified in queries using the COM input format is delivered as-is to the custom input format COM object as an argument to the <u>OpenInput</u> method of the <u>ILogParserInputContext</u> interface, and its syntax and interpretation is provided by the custom input format selected. The <<u>from-entity</u>> specified in queries using the COM input format must however obey the <u>general syntax</u> for <<u>from-entity</u>> language elements.

COM Input Format Fields

The input records generated by the COM input format contain the fields provided by the currently selected Custom Input Format COM plugin.

The number of fields, their names, and their data types are retrieved through the <u>GetFieldCount</u>, <u>GetFieldName</u>, and <u>GetFieldType</u> methods of the <u>ILogParserInputContext</u> interface.

COM Input Format Parameters

The COM input format supports the following parameters:

iProgID

Values:	COM ProgID
Default:	not specified
Description:	ProgID of the Custom Input Format COM Plugin.
Details:	This parameter is used to specify the version- independent ProgID of the custom input format COM object selected for the current query.
Example:	-iProgID:MSUtil.LogQuery.Sample.QFE

iCOMParams

Values:	name=value[,name=value]
---------	-------------------------

Default: not specified

Description: Parameters for the Custom Input Format COM Plugin.

Details: The value of this parameter is a comma-separated list of name-value pairs specifying property names and values for Custom Input Format COM Plugins implemented through the IDispatch COM interface. If property names or their values contain space characters, the value of this parameter should be surrounded by double-quote (") characters. For more information on custom properties exposed by COM plugins, see <u>Custom Properties</u> in the <u>COM Input</u> <u>Format Plugins reference</u>.

Example: -iCOMParams:TargetMachine=localhost,ExtendedFields=on

iCOMServer

Values: computer name

Default: localhost

- Description: Computer name on which the Custom Input Format COM Plugin is to be instantiated.
- Details: Plugin COM objects supporting Distributed COM (DCOM) can be instantiated on a remote computer, thus providing a means for the custom input format to process data on a computer different than the computer running the Log Parser query.

Example: -iCOMServer:MYSERVER01

COM Input Format Examples

QFE Information

Return QFE information from the local machine, using the "QFE" sample Custom Input Format COM Plugin:

LogParser "SELECT * FROM ." -i:COM -iProgID:MSUtil.LogQuery.Sample. QFE -iCOMParams:ExtendedFields=on

CSV Input Format

The CSV input format parses comma-separated values text files.

CSV text files are generated and handled by a large number of applications and tools, including:

- Microsoft Excel
- PerfMon
- Generic spreadsheet applications

In a CSV text file, each line consists of one record, and fields in a record are separated by commas.

Depending on the application, the first line in a CSV file might be a "header", containing the labels of the record fields.

The following example shows a CSV file beginning with a header:

DateTime, PID, Comment

5/28/2004 13:56:12, 2956, Application started

5/28/2004 15:24:42, 1048, Application started

"\\GAB1\Processor(_Total)\% Processor Time","\\GAB1\System\Processes"

"99.999993086289507","33" "2.0000000000000018","33" "1.000000000000009","33" "0.33333333333332993","33" "0.333333333333332993","33" "0.3333333333333332993","33" "0.33333333333333332993","33" "4.0000000000000036","33" "4.3333333333333333333339","33"

See also:

<u>TSV Input Format</u> <u>CSV Output Format</u>

CSV Input Format From-Entity Syntax

<from-entity> ::= <filename> [, <filename> ...] | http://<url> | STDIN

The <<u>from-entity></u> specified in queries using the CSV input format is either:

- A comma-separated list of paths of CSV files, eventually including wildcards;
- The URL of a file in the CSV format;
- The "STDIN" keyword, which specifies that the input data is available from the input stream (commonly used when piping command executions).

Examples:

FROM LogFiles1*.csv, LogFiles2*.csv, \\MyServer\FileShare*.csv

FROM http://www.microsoft.adatum.com/MyCSVFiles/example.csv

type data.csv | LogParser "SELECT * FROM STDIN" -i:CSV

CSV Input Format Fields

The structure of the input records generated by the CSV input format is determined at run time, depending on the data being parsed, and on the values specified for the input format parameters.

The first two input record fields are fixed, and they are described in the following table:

Name	Туре	Description
Filename	STRING	Full path of the file containing this entry
RowNumber	INTEGER	Line in the file containing this entry

These two fields are then followed by the fields detected by the CSV input format in the CSV file(s) being parsed. The number, names, and data types of the fields are determined by examining initially the CSV data according to the values specified for the input format parameters.

The number of fields detected by the CSV input format during the initial inspection phase dictates how the CSV record fields will be extracted from the input data during the subsequent parsing stage.

If a CSV line contains less fields than the number of fields established, the missing fields are returned as NULL values.

On the other hand, if a CSV line contains more fields than the number of fields established, the extra fields are parsed as if they were part of the value of the last field expected by the CSV input format.

Number of Fields

The number of fields in an input record is determined by the input CSV data and by the values of the <u>nFields</u> and <u>fixedFields</u> parameters.

When the "nFields" parameter is set to -1, the CSV input format determines the number of fields by inspecting the input CSV data.

If the "fixedFields" parameter is set to "ON", indicating that all the rows in the CSV file have the same fixed number of fields, then the number of fields is determined by parsing either the first line of the CSV input data, or the first line of the header file specified with the "iHeaderFile" parameter.

On the other hand, if the "fixedFields" parameter is set to "OFF", indicating that the rows in the CSV file have a variable number of fields, then the number of fields is assumed to be the largest number of fields found among the first *n* lines of the CSV input data (eventually including the first line of the header file specified with the "iHeaderFile" parameter), where *n* is the value of the "dtLines" parameter.

As an example, the following CSV file contains a variable number of fields:

Name, City, AreaCode

Jeff, Redmond, 425

Vshere, paraged, with, the 'n Fields' parameter set to -1 and the "fixed Fields" parameter set to -1 and the "f

In this case, the extra fourth field in the second record would be parsed as part of the third "AreaCode" field, whose value would then be "206, 98101".

On the other hand, if the "fixedFields" parameter is set to "OFF", and the "dtLines" parameter is set to any value greater than 2, then the same CSV file would yield four fields ("Name", "City", "AreaCode", and an additional fourth field detected in the second CSV record).

In this case, the first and third records would have a NULL value for the fourth field, and the second record would have a "98101" value for the fourth field.

When the "nFields" parameter is set to a value greater than zero, the CSV input format uses the specified value as the number of fields in the input data.

However, if the "fixedFields" parameter is set to "OFF", indicating that the rows in the CSV file have a variable number of fields, then the CSV input format uses the value of the "nFields" parameter as a "suggested minimum" number of fields, and it examines the first *n* lines of the CSV

input data (eventually including the first line of the header file specified with the "iHeaderFile" parameter), where n is the value of the "dtLines" parameter, to determine the number of fields among these lines. If lines are found containing more fields than the value specified for the "nFields" parameter, then the number of fields is adjusted to the largest number of fields found among the first n lines.

Considering again the previous CSV example file, parsing the file with the "nFields" parameter set to 3 and the "fixedFields" parameter set to "ON" would yield three fields.

However, setting the "fixedFields" parameter to "OFF" and the "dtLines" parameter to any value greater than 2 would yield four fields, detecting the extra field in the second record.

Field Names

The names of the fields in an input record is determined by the input CSV data and by the values of the <u>headerRow</u> and <u>iHeaderFile</u> parameters.

When the "headerRow" parameter is set to "ON", the CSV input format assumes that the first line in the CSV file being parsed is a header containing the field names.

In this case, if the "iHeaderFile" parameter is left unspecified, the CSV input format extracts the field names from the header line.

On the other hand, if the "iHeaderFile" parameter is set to the path of a CSV file containing at least one line, then the CSV input format assumes that the specified file contains a header, parses its first line only, and extracts the field names from this line, ignoring the first line of the CSV file being parsed.

If the number of field names extracted is less than the number of fields detected, the additional fields are automatically named "FieldN", with N being a progressive index indicating the field position in the input record.

Considering the previous example CSV file, setting the "headerRow" parameter to "ON" would cause the CSV input format to use the first line of the CSV file as a header containing the field names.

With the "fixedFields" parameter set to "ON", the CSV input format would detect three fields, whose names would be "Name", "City", and

"AreaCode".

On the other hand, with the "fixedFields" parameter set to "OFF", the CSV input format would detect four fields, named "Name", "City", "AreaCode", and "Field4".

When the "headerRow" parameter is set to "OFF", the CSV input format assumes that the CSV file being parsed does not contain a header, and that its first line is the first data record in the file.

In this case, if the "iHeaderFile" parameter is set to the path of a CSV file containing at least one line, then the CSV input format assumes that the specified file contains a header, parses its first line only, and extracts the field names from this line.

On the other hand, if the "iHeaderFile" parameter is left unspecified, the fields are automatically named "Field*N*", with *N* being a progressive number indicating the field position in the input record.

As an example, the following CSV file does not contain a header line:

Jeff, Redmond, 425

Steve, Seattle, 206

When parsed with the "headerRow" parameter to "OFF", the CSV input format assumes that the first line of the CSV file is the first data record in the file. In this case, the three fields would be named "Field1", "Field2", and "Field3".

Field Types

The <u>data type</u> of each field extracted from the input data is determined by examining the first n CSV data lines, where n is the value specified for the <u>dtLines</u> parameter, in the following way:

- If all the non-empty field values in the first *n* lines are formatted as decimal numbers, then the field is assumed to be of the <u>REAL</u> type.
- If all the non-empty field values in the first *n* lines are formatted as integer numbers, then the field is assumed to be of the <u>INTEGER</u> type.
- If all the non-empty field values in the first *n* lines are formatted as timestamps in the format specified by the <u>iTsFormat</u> parameter, then the field is assumed to be of the <u>TIMESTAMP</u> type.
- Otherwise, the field is assumed to be of the <u>STRING</u> type.

Empty field values are returned as <u>NULL</u> values.

CSV Input Format Parameters

The CSV input format supports the following parameters:

headerRow

Values:	ON OFF
Default:	ON
Description:	Specifies whether or not the input CSV file(s) begin with a header line.
Details:	When this parameter is set to "ON", the CSV input format assumes that each file being parsed begins with a header line, containing the labels of the fields in the file. If the "iHeaderFile" parameter is left unspecified, the CSV input format will use the field names in the first file's header as the names of the input record fields. If a value is specified for the "iHeaderFile" parameter, the CSV input format will ignore the header line in each file being parsed. When this parameter is set to "OFF", the CSV input format assumes that the file(s) being parsed do not contain a header, and parses their first line as data records. For more information on headers and field names, see <u>CSV Input Format Fields</u> .
Example:	-headerRow:OFF

iHeaderFile

Values:	path to a CSV file
---------	--------------------

Default: *not specified*

Description: File containing field names.

Details: When parsing CSV files that do not contain a header

line, the fields of the input records produced by the CSV input format are named "Field1", "Field2", ... To override this behavior and use meaningful field names, this parameter can be set to to the path of a CSV file containing a header line, causing the CSV input format to use the field names in the specified CSV file's header line as the names of the input record fields. Only the first line of the specified CSV file is parsed, and eventual additional lines are ignored. For more information on headers and field names, see <u>CSV Input Format Fields</u>.

Example: -iHeaderFile:"C:\My Folder\header.csv"

fixedFields

- Values: **ON | OFF**
- Default: ON
- Description: Specifies whether or not all the records in the input CSV file(s) have a fixed number of fields.

Details: When this parameter is set to "ON", the CSV input format assumes that the number of fields in all the input CSV records equals the number of fields found in the first CSV line parsed, or the number of fields specified for the "nFields" parameter.
When this parameter is set to "OFF", the CSV input format assumes that the input CSV records have a variable number of fields, and it parses the first *n* lines of the input CSV data to determine the maximum number of fields in the records, where *n* is the value specified for the "dtLines" parameter.
For more information on how the number of fields is determined, see CSV Input Format Fields.

Example: -fixedFields:OFF

nFields

Values:	number of fields (number)
---------	---------------------------

Default: -1

Description: Number of fields in the CSV data records.

Details: When the "fixedFields" parameter is set to "ON", this parameter specifies the number of fields in the input CSV data.

When the "fixedFields" parameter is set to "OFF", this parameter specifies the minimum number of fields in the input CSV data. If the first *n* lines of input data contain more fields than the specified number of fields, where *n* is the value of the "dtLines" parameter, then the number of fields is assumed to be the maximum number of fields found within the *n* lines of data. The special "-1" value specifies that the number of fields is to be deducted by inspecting the first *n* lines of input data, where *n* is the value of the "dtLines" parameter.

determined, see <u>CSV Input Format Fields</u>.

Example: -nFields:3

dtLines

- Values: number of lines (number)
- Default: 10
- Description: Number of lines examined to determine number of fields and field types at run time.
- Details: This parameter specifies the number of initial lines that the CSV input format examines to determine the number of the input record fields and the data type of each field.

If the value is 0, all fields will be assumed to be of the <u>STRING</u> data type.

For more information on how the number of fields and their data types are determined, see <u>CSV Input Format</u> <u>Fields</u>.

Example: -dtLines:50	
----------------------	--

iDQuotes

- Values: Auto | Ignore
- Default: Auto

Description: Behavior with double-quoted fields.

Details: When this parameter is set to "Auto" and a field value is enclosed within double-quote characters ("), the CSV input format parses the field ignoring comma characters (,) within the double-quotes, and returns the enclosed value stripping off the surrounding double-quote characters.
When set to "Ignore", the CSV input format does not perform any double-quote processing, and field values are returned verbatim, including double-quote characters.

Example:

-iDQuotes:Ignore

nSkipLines

- Values: number of lines (number)
- Default: 0

Description: Number of initial lines to skip.

Details: When this parameter is set to a value greater than zero, the CSV input format skips the first *n* lines of each input file before parsing its header line, where *n* is the value specified for this parameter. Example: -nSkipLines:5

comment

Values:	any string
Default:	not specified
Description:	Skip lines beginning with this string.
Details:	When this parameter is set to a non-empty string, the CSV input format skips all the input CSV lines that begin with this string.
Example:	-comment:"Meta Data:"

iCodepage

Values:	codepage ID (number)
Default:	0
Description:	Codepage of the CSV file.
Details:	0 is the system codepage, -1 is UNICODE.
Example:	-iCodepage:1245

iTsFormat

Values:	timestamp format
Default:	yyyy-MM-dd hh:mm:ss
Description:	Format of timestamp values in the input CSV data.
Details:	This parameter specifies the date and/or time format used in the CSV data being parsed. Values of fields matching the specified format are returned as values of the <u>TIMESTAMP</u> data type. For more information on date and time formats, see <u>Timestamp Format</u>

Specifiers.

Example:	-iTsFormat:"MMM dd, yyyy"

iCheckpoint

Values:	checkpoint filename
Default:	not specified
Description:	Load and save checkpoint information to this file.
Details:	This parameter enables the "Incremental Parsing" feature that allows sequential executions of the same query to only process new events that have been logged since the last execution. For more information, see <u>Parsing Input Incrementally</u> .
Example:	-iCheckpoint:C:\Temp\myCheckpoint.lpc

CSV Input Format Examples

Average Processor Usage per Minute

Parse a PerfMon CSV log file and calculate the average processor usage per minute:

LogParser "SELECT QUANTIZE([(PDH-CSV 4.0) (Pacific Daylight Time)(4 20)], 60) AS Minute, AVG([\\GAB1\Processor(_Total)\% Processor Time]) AS AVGProcessor FROM PerfMon_000001.csv GROUP BY Minute" -i:CSV -iT _sFormat:"MM/dd/yyyy hh:mm:ss.ll"

ETW Input Format

The ETW input format parses Enterprise Tracing for Windows trace log files (.etl files) and live ETW trace sessions.

Enterprise Tracing for Windows (ETW) is a framework for implementing tracing providers that can be used for debugging and capacity planning. An ETW trace log or live session consists of a stream of "Events", each published by a "Provider". Windows event providers include the Kernel, IIS, COM+, and many other Windows components.

Each event has its own set of named properties, or fields, containing the event data. The structure of each event is described by a WMI class derived from the "EventTrace" class and registered with the WMI repository during the setup of the provider component. The ETW input format queries the WMI repository for these classes in order to retrieve information about the structure of each event.

ETW trace log files and live sessions can be controlled through either the PerfMon utility, or through the tracelog.exe or logman.exe command-line tools.

From-Entity Syntax Fields Parameters Examples

ETW Input Format From-Entity Syntax

<from-entity> ::= <etl_file_name> [, <etl_file_name> ...] | <live_session_name>

The <from-entity> specified in queries using the ETW input format can assume one of the following values:

- A comma-separated list of paths to .etl ETW trace log files;
- The name of an ETW live tracing session.

Examples:

FROM MyTrace1.etl, MyTrace2.etl, MyTrace3.etl

FROM \\COMPUTER01\TraceFiles\MyTrace.etl, \\COMPUTER02\TraceFiles \MyTrace.etl

FROM MyLiveSession

ETW Input Format Fields

The structure of the input records generated by the ETW input format is determined at run time, depending on the ETW trace being parsed, and on the value specified for the <u>fMode</u> ("field mode") parameter, which can be set to "Compact", "FNames", "Full", or "Meta".

Compact Field Mode

When the "fMode" parameter is set to "Compact", the ETW input format generates an input record for each event in the trace being parsed. In this mode, input records contain four fields common to all the events, plus an additional "UserData" field containing the values of all the properties specific to the event being processed, concatenated into a single string value using the character specified for the <u>compactModeSep</u> parameter as a separator between the values.

The following table shows the fields of the input records generated in the "Compact" field mode:

Name	Туре	Description
EventNumber	INTEGER	Index of this event in the trace being parsed
EventName	STRING	Name of the event
EventTypeName	STRING	Name of the event type
Timestamp	TIMESTAMP	Date and time at which the event was traced
UserData	STRING	Event-specific property values

The following example shows some sample "UserData" field values

generated in the "Compact" field mode:

UserData

Thef" Genpartölfield prodeantestides and reasily readable way to display the events contained in an 157W trace doutogy pries operating in this mode can not reference of opparties operating in this mode can

FNames Field Mode

The "FNames" field mode operates similar to the "Compact" field mode, but each property value in the "UserData" field is preceded by the name of the property for better readability.

The following example shows some sample "UserData" field values generated in the "FNames" field mode:

UserData

AppPoolId=DefaultAppPool|RawConnId=0|RequestURL=http://localhost:80/| RequestVerb=GET

Follin Field Moor 0000-0000-0000-1200-006000000fc} | RequestURL =/ In Amp policita Default, App Power and the states and the default of the states of the st

In this mode, input records contain a field for each property of each event generated by the providers in the trace being parsed.

When operating in "Full" field mode, the ETW input format works with a two-stage approach.

During the first stage, the ETW input format examines the input trace to determine which providers have logged events in the trace being parsed. When the providers parameter is left unspecified, the ETW input format pre-processes a number of events equal to the value specified for the <u>dtEventsLog</u> or <u>dtEventsLive</u> parameters, depending on whether or not the trace being parsed is a trace log file or a live trace session. After parsing these initial events, the ETW input format assumes that the trace

being parsed contains all the events that can be logged by the providers found among these initial events.

On the other hand, when the "providers" parameter is set to either a comma-separated list of provider names or GUIDs or to the path to a text file containing a list of provider names or GUIDs, the ETW input format assumes that the trace being parsed contains all the events that can be logged by the specified providers.

Once the set of providers logging in the input trace has been identified, the ETW input format "constructs" the input record structure. The first 20 input record fields are common to all the events, and they are described in the following table:

Name	Туре	Description
TraceName	STRING	Trace file or session name containing this event
EventNumber	INTEGER	Index of this event in the trace being parsed
Timestamp	TIMESTAMP	Date and time at which the event was traced
InstanceID	INTEGER	InstanceID field of this event
ParentInstanceID	INTEGER	ParentInstanceID field of this event
ParentGUID	STRING	ParentGUID field of this event
ProviderDescription	STRING	Name of the provider of this event

ProviderGUID	STRING	GUID of the provider of this event
EventName	STRING	Name of this event
EventDescription	STRING	Description of this event
EventVersion	INTEGER	Version of this event
EventGUID	STRING	GUID of this event
EventType	INTEGER	Type of this event
EventTypeName	STRING	Name of this event type
EventTypeDescription	STRING	Description of this event type
EventTypeLevel	INTEGER	Level of this event type
ThreadID	INTEGER	ID of the thread that logged this event
ProcessID	INTEGER	ID of the process that logged this event
KernelTime	INTEGER	Elapsed execution time for kernel mode instructions, in CPU ticks
UserTime	INTEGER	Elapsed execution time for user mode instructions, in CPU ticks

These 20 fields are then followed by the *union* of all the properties of all

the events that can be logged by the providers identified during this stage.

During the second stage, the ETW input format parses the trace events from beginning to end, generating an input record for each event. For any given event, only the first 20 input record fields and the fields corresponding to the event properties are populated with a value; all the other input record fields corresponding to properties of other events are set to NULL values.

The following sample output shows selected fields from the input records generated when parsing the previous example in "Full" field mode:

AppPoolId	RawConnId ContextId	RequestURL	
RequestV	/erb		
Queries operating in "Full" mode can refer to individual properties of			of
events, but the input records generated contain too many fields for the			
resultate preaily redable.		http://localhost:80/	GE
Т			
		000 000000000000000000000000000000000	

- - {0000000-0000-1200-006000000fc} /

Meta Field Mode

DefaultAppPool 0 - http://localhost:80/default.htm Inc Meta" field mode, the ETW input format returns meta-information about events, generating an input record for each property of each event that can be logged by each provider in the trace(s) being parsed. Input records contain meta-data about the event properties, including information about the property type, information about the event containing the property, and information about the provider generating the event.

The "Meta" field mode employs a two-stage parsing schema similar to the "Full" field mode. During the first stage, the ETW input format preprocesses the input trace to determine the set of providers that generated events in the trace.

In this mode, however, once the set of providers has been identified, the ETW input format does not process the trace, but rather returns the event meta-information populating the input record fields described in the following table:

Name	Туре	Description
ProviderDescription	STRING	Description of the provider
ProviderClassName	STRING	WMI class name of the provider
ProviderGUID	STRING	GUID of the provider
EventName	STRING	Name of the event
EventDescription	STRING	Description of the event
EventVersion	INTEGER	Version of the event
EventClassName	STRING	WMI class name of the event
EventGUID	STRING	GUID of the Event
EventType	INTEGER	Type of the event
EventTypeName	STRING	Name of the event type
EventTypeDescription	STRING	Description of the event type
EventTypeClassName	STRING	WMI class name of the event type
EventTypeLevel	INTEGER	Level of the event type
FieldName	STRING	Name of this event field
FieldDescription	STRING	Description of this event field
FieldIndex	INTEGER	Index of this field among the event's fields

ETW Input Format Parameters

The EVT input format supports the following parameters:

fMode

Values:	Full Compact FNames Meta
Default:	FNames
Description:	Operation mode.
Details:	This parameter specifies how the ETW input format should return the information contained in the trace(s) being parsed. For more information on the different field modes, see <u>ETW Input Format Fields</u> .
Example:	-fMode:Full

providers

Values:	filename or comma-separated list of provider names or GUIDs
Default:	not specified
Description:	List of providers for the "Full" or "Meta" field modes.
Details:	This parameter specifies the set of providers logging to the input trace(s) to allow the "Full" or "Meta" field modes to early detect the providers to process. The value of this parameter can either by the path to a text file containing the providers' GUIDs (in the same format accepted by the "pf" argument of the logman.exe tool), or a comma-separated list of provider names or GUIDs. If this parameter is not specified when the ETW input format operates in "Full" or "Meta" field mode, then the set of providers will be detected by pre-processing the first <i>n</i> events, where <i>n</i> is the value specified for the

"dtEventsLog" or "dtEventsLive" parameters. For more information about the different field modes, see <u>ETW Input Format Fields</u>.

Examples: -providers:MyProviders.guid -providers:"IIS: WWW Server,IIS: Active Server Pages (ASP)"

dtEventsLog

Values: number of events (number)

Default: 3000

- Description: Number of trace log file events examined to detect the set of providers in "Full" or "Meta" field modes.
- Details: This parameter specifies the number of initial events that the ETW input format examines to detect the set of providers logging in an input trace log file when operating in the "Full" or "Meta" field modes. The value of this parameter is only used when the "providers" parameter is left unspecified. For more information about the different field modes, see <u>ETW Input Format Fields</u>.

Example: -dtEventsLog:100

dtEventsLive

Values: number of events (number)

Default: 20

- Description: Number of live trace session events examined to detect the set of providers in "Full" or "Meta" field modes.
- Details: This parameter specifies the number of initial events that the ETW input format examines to detect the set of providers logging in an input live trace session when
operating in the "Full" or "Meta" field modes. The value of this parameter is only used when the "providers" parameter is left unspecified. For more information about the different field modes, see <u>ETW Input Format Fields</u>.

Example: -dtEventsLive:100

flushPeriod

Values: milliseconds

Default: 500

- Description: Number of milliseconds between live trace session flushes.
- Details: When processing a live trace session, the internal buffering mechanisms of the ETW infrastructure might cause events to appear with a noticeable delay. This parameter specifies how often the ETW input format should force a buffer flush to retrieve real-time events.

Example: -flushPeriod:2000

ignoreEventTrace

Values:	ON OFF
Default:	ON
Description:	Ignore EventTrace events.
Details:	The very first event in any trace session is the "EventTrace" event, which contains meta-data about the trace session. This parameter specifies whether or not this event should be processed and returned by the ETW input format.
Example:	-ignoreEventTrace:OFF

compactModeSep

Values:	any string
Default:	I
Description:	Separator between the values of the "UserData" field in the "Compact" or "FNames" field modes.
Details:	When operating in the "Compact" or "FNames" field modes, the "UserData" field contains all the properties of the event being processed concatenated one after the other, using the value of this parameter as a separator between the elements.
Example:	-compactModeSep:,

expandEnums

Values:	ON OFF
Default:	ON
Description:	Expand enumeration event properties.
Details:	Many ETW events contain numeric properties whose values describe enumerations. This parameter specifies whether or not the numeric values of properties of this type should be expanded to return the text representation of the enumeration values.
Example:	-expandEnums:OFF

ignoreLostEvents

Values:	ON OFF
Default:	ON
Description:	Ignore lost events.

Details: ETW traces contain information about events that might have been lost during the tracing session. If this parameter is set to "OFF" and the input trace indicates the presence of lost events, the ETW input format generates a <u>warning</u> when the trace has been completely processed showing the number of events that have been lost.

Example: -ignoreLostEvents:OFF

schemaServer

Values:	computer name
Default:	not specified
Description:	Name of computer with event schema information.
Details:	This parameter specifies the name of the computer whose WMI repository contains the schema information for the events being parsed. When this parameter is not specified, the ETW input format connects to the computer specified in the <u>from-</u> <u>entity</u> if parsing a trace file from a remote computer, or to the local computer if parsing a local trace file or live tracing session.
Example:	-schemaServer:MYCOMPUTER02

ETW Input Format Examples

Parsing an IIS 6.0 ETW Trace Log File

This example shows how to start a trace session containing events from the IIS 6.0 providers, how to stop the session, and how to parse the resulting trace log file.

The example commands shown here apply to Windows Server 2003.

1. List the GUIDs of the providers registered with the system using the following command from a command-line window:

C:\>logman query providers The output of this command will look like the following sample: Provider GUID 2. Identify the providers needed for the the see set out din soise-13 example, the trace session will be enabled for the "IIS: WWW Server Dandr" HS: Aptive: Server Pagesa 60 SP2) - 2009 - 2010 - 2 pActivity Directorye, Kerlowed by the that addited set of the the set of the 167.eeb6966besc40 r the provider. For more information on the allaid the flags and levels for alphovider, coopstell the 0e-0dbd61 co6180bent documentation. The: following Nexa in the shows a test 7 to branched 1-44b4-a95e-3c "MED to defre s.guid" containing the "IIS: WWW Server" and "ITS:: Active/Server Pages (ASP)"{Brownittens:4c21-4981-ae10-3f da0d9b0f83HSi2A4tion-Server Dogese (ASP) a0d9b of 8509 0 d part fr 15 g-456e-a4ef

{065994492cb45}e-456e-a4ef-37c984a2cb4b} 0xffffffff 54. StantalheetnatingAsetssion(UsiA) the provide982fexb41e1as12h924a4g0060082ff90@1'}-pf" logman command-line parameter:IIS: IISADMIN Global{DC1271C2-A0AF-400f-850



EventNumber EventName EventTypeName Tim

Timestamp

UserData

Parsing a live IIS 6.0 ETW-Trace-Session ------

This example shows how to start a live trace session containing events from the IIS 6.0 providers, how to start a Log Parser command that shows the events in real-time, and how to stop the session. The example commands shown here apply to Windows Server 2003. -14 20:27:26.624.399000 ContextId={00000000-0000-0000-1200

- 1. Execute steps: 1 Site In the example Default AppPool ConnId=-2
- Lxecute steps 1-5 nom the example above. 11 88230375077969904 RawConnId=0 RequestURL=http://localhost start the tracing session using the providers text file as the adv/Request Verb=GET argument of the "-of" logman command-line parameter. 3900 Start FILTER START specifying also the "-nt" flag to enable a real-time tracing:27:26 624.399000 ContextId={00000000-0000-0000-1200-0060000000 session: fc}|FilterName=C:\WINNT\Microsoft.NET\Framework\v1.1.4322
 \aspnet_filter.dll C:\>logman start Example Trace of MyProviders guid-ets rt 2004-10-1420:27:26 Specifying also the "-nt" flag to enable a real-time tracing: fc}|FilterName=C:\WINNT\Microsoft.NET\Framework\v1.1.4322

10-14 20:27:26.624.399000 ContextId={00000000-0000-0000-12

- 5. TODE ODEGCIO ODE Stight has now started, and the selected provider \$150 All the ID Join for epender BOC each Apprest Fon the 20594-1 V0eb4 Stor 2726.624.399000 ContextId={00000000-0000-120
- 6. Flora & Separate command-line shell window, execute the following ISE Parser Torm fand to parse the following ISE in the following ISE is the session of the following ISE is the session of the sector of the session of the sessio

C:\>LogParser*SELECP*START Example Trace*-1:ETW .624.399000 ContextId={00000000-0000-0000-1200-0060000000

This Files Marser Conneared Files Generate Files Microne f

7. When desired, the tracing session can be stopped with the 12 following command:

9 IISFilter FILTER_PREPROC_HEADERS_END 2004-1 0-14 20:27:26.624.399000 ContextId={00000000-0000-0000-120 C:>logman stop Example Trace -ets 0-0060000000fc}

11 IISCache URL_CACHE_ACCESS_START 2004-10 -14 20:27:26.624.399000 ContextId={00000000-0000-0000-1200 -006000000fc}|RequestURL=/

EVT Input Format

The EVT input format returns events from the Windows Event Log and from Event Log backup files (.evt files).

This input format reads event information from the Windows Event Log, including local and remote System, Application, Security, and custom event logs, as well as from Event Log backup files.

<u>From-Entity Syntax</u> <u>Fields</u> <u>Parameters</u> <u>Examples</u>

EVT Input Format From-Entity Syntax

<from-entity></from-entity>	::=	<event_log> [, <event_log>]</event_log></event_log>
<event_log></event_log>	::=	[\\ <computer_name>\]<event_log_name> </event_log_name></computer_name>
		<event_log_backup_filename></event_log_backup_filename>

The <<u>from-entity</u>> specified in queries using the EVT input format is a comma-separated list of:

- Names of Event Logs ("System", "Application", "Security", or a custom event log), optionally preceded by the name of a remote computer in the UNC notation;
- Paths of Event Log backup files (.evt files), optionally including wildcards.

Names of custom event logs that include space characters must be specified within single-quote characters.

Examples:

FROM System, Application, \\SERVER2\System, \\SERVER2\Application

FROM System, Application, 'My Custom Event Log'

FROM D:\MyEVTLogs*.evt, \\SERVER2\D\$\MyEVTLogs*.evt

FROM System, D:\MyEVTLogs\System.evt

EVT Input Format Fields

The input records generated by the EVT input format contain the following fields:

Name	Туре	Description
EventLog	STRING	Name of the Event Log or Event Log backup file containing this event
RecordNumber	INTEGER	Index of this event in the Event Log or Event Log backup file containing this event
TimeGenerated	TIMESTAMP	The date and time at which the event was generated (local time)
TimeWritten	TIMESTAMP	The date and time at which the event was logged (local time)
EventID	INTEGER	The ID of the event
EventType	INTEGER	The numeric type of the event
EventTypeName	STRING	The descriptive type of the event
EventCategory	INTEGER	The numeric category of the

		event
EventCategoryName	STRING	The descriptive category of the event
SourceName	STRING	The source that generated the event
Strings	STRING	The textual data associated with the event
ComputerName	STRING	The name of the computer on which the event was generated
SID	STRING	The Security Identifier associated with the event
Message	STRING	The full event message
Data	STRING	The binary data associated with the event

EVT Input Format Parameters

The EVT input format supports the following parameters:

fullText

Values:	ON OFF
Default:	ON
Description:	Retrieve the full text message.
Details:	This parameter enables/disables the retrieval of Event Lomessages.
Example:	-fullText:OFF

resolveSIDs

Values:	ON OFF
Default:	OFF
Description:	Resolve SID values into full account names.
Details:	When set to "ON", this parameter causes the EVT input perform an account name lookup for each SID value in the parsed, and return the account name instead of the SID value.
Example:	-resolveSIDs:ON
matMsø	

formatMsg

Values:	ON OFF
Default:	ON
Description:	Format the text message as a single line.
Details:	Event text messages often span multiple lines. When thi

set to "ON", the EVT input format preserves readability o by removing carriage-return, line-feed, and multiple spac from the message text.

When this parameter is set to "OFF", the EVT input form original message text with no intervening post-processin

Example: -formatMsg:OFF

msgErrorMode

- Values: NULL | ERROR | MSG
- Default: MSG
- Description: Behavior when event messages or event category name resolved.
- Details: The text of an event log message and the textual name (are stored in binary files installed with the application tha event log. In some cases, uninstalling the application or I the application might cause the loss of the necessary bir making it impossible to retrieve the text data for those ev been logged prior to the reconfiguration.

This parameter specifies the desired behavior for the EV when an event log message text or its category name ca retrieved.

When this parameter is set to "NULL", the "Message" or "EventCategoryName" field value is returned as a NULL to "ERROR", a <u>parse error</u> is returned. When set to "MS message is returned for the field, specifying that the text or the category name could not be found.

Example: -msgErrorMode:NULL

fullEventCode

Values:	ON OFF
Default:	OFF

Description:	Return the full event ID code instead of the friendly code
Detailer	When this parameter is set to " ON " the EV/T input forma

Details: When this parameter is set to "ON", the EVT input forma 32-bit value of the event ID code. When set to "OFF", the format returns the lower 16-bit value of the code (as disp Event Viewer).

Example: -	fullEventCode:ON
------------	------------------

direction

Values:	FW BW
Default:	FW
Description:	Chronological direction in which events are retrieved.
Details:	When set to "FW", events are retrieved from the oldest to When set to "BW", events are retrieved from the newest This parameter is especially useful with queries that use keyword to retrieve the last <i>n</i> logged events.
Example:	-direction:BW
stringsSep	
Values:	any string

Description: Separator between values of the "Strings" field.

Details: The "Strings" field contains an *array* of text data associat event. The value of this field is built by concatenating the one after the other, using the value of this parameter as a between the elements.

Example: -stringsSep:,

iCheckpoint

Default:

I

- Values: checkpoint filename
- Default: not specified

Description: Load and save checkpoint information to this file.

Details: This parameter enables the "Incremental Parsing" featur sequential executions of the same query to only process have been logged since the last execution. For more infor <u>Parsing Input Incrementally</u>.

Example: -iCheckpoint:C:\Temp\myCheckpoint.lpc

binaryFormat

- Values: ASC | PRINT | HEX
- Default: HEX

Description: Format of the "Data" binary field.

Details: The "Data" field contains binary data that is often not suit textually represented. When this parameter is set to "ASC", data bytes belongin 0x7F range are returned as ASCII characters, while data the range are returned as period (.) characters, as shown example:

Bucket: 02096553..rundll32.exe

When this parameter is set to "PRINT", data bytes repres ASCII characters are returned as ASCII characters, while do not represent printable ASCII characters are returned characters, as shown in the following example:

Bucket: 02096553

rundll32.exe

When this parameter is set to "HEX", all data bytes are r digit hexadecimal values, as shown in the following exan 4275636B65743A2030323039363535330D0A72756E646C(

Example: -binaryFormat:PRINT

EVT Input Format Examples

Logons

Create an XML report file containing logon account names and dates from the Security Event Log:

LogParser "SELECT TimeGenerated AS LogonDate, EXTRACT_TOKEN(Str ings, 0, '|') AS Account INTO Report.xml FROM Security WHERE EventID N OT IN (541;542;543) AND EventType = 8 AND EventCategory = 2"

Event Distribution

Retrieve the distribution of EventID values for each Event Source:

LogParser "SELECT SourceName, EventID, MUL(PROPCOUNT(*) ON (So urceName), 100.0) AS Percent FROM System GROUP BY SourceName, Eve ntID ORDER BY SourceName, Percent DESC"

Event Message Report

Create TSV files containing Event Messages for each Source in the Application Event Log:

LogParser "SELECT SourceName, Message INTO myFile_*.tsv FROM \\MY SERVER1\Application, \\MYSERVER2\Application"

FS Input Format

The FS input format returns information on files and directories.

The FS input format enumerates the files and directories matching the search path(s) specified in the <u>from-entity</u>, much like the Windows shell "dir" command, returning an input record for each file and directory in the enumeration.

<u>From-Entity Syntax</u> <u>Fields</u> <u>Parameters</u> <u>Examples</u>

See also:

REG Input Format

FS Input Format From-Entity Syntax

<from-entity> ::= <path> [, <path> ...]

The <u><from-entity></u> specified in queries using the FS input format is a comma-separated list of paths, eventually containing wildcards.

Examples:

FROM C:\Windows*.dll, \\MYSERVER\C\$\Windows*.dll

FROM *.*

FROM C:*.*, D:*.*

FROM C:\Windows\Explorer.exe

FS Input Format Fields

The input records generated by the FS input format contain the following fields:

Name	Туре	Description
Path	STRING	Full path of the file or directory
Name	STRING	Name of the file or directory
Size	INTEGER	Size of the file, in bytes
Attributes	STRING	Attributes of the file or directory
CreationTime	TIMESTAMP	Date and time at which the file or directory has been created (local or UTC time, depending on the value of the <u>useLocalTime</u> parameter)
LastAccessTime	TIMESTAMP	Date and time at which the file or directory has been last accessed (local or UTC time, depending on the value of the <u>useLocalTime</u> parameter)
LastWriteTime	TIMESTAMP	Date and time at which the file or directory has been last modified (local or UTC time, depending on the value of the <u>useLocalTime</u> parameter)

<u> </u>	l <u></u>	
FileVersion	STRING	Version of the file
ProductVersion	STRING	Version of the product the file is distributed with
InternalName	STRING	Internal name of the file
ProductName	STRING	Name of the product the file is distributed with
CompanyName	STRING	Name of the vendor company that produced the file
LegalCopyright	STRING	Copyright notices that apply to the file
LegalTrademarks	STRING	Trademarks and registered trademarks that apply to the file
PrivateBuild	STRING	Private version information of the file
SpecialBuild	STRING	Special file build notes
Comments	STRING	Comments associated with the file
FileDescription	STRING	Description of the file
OriginalFilename	STRING	Original name of the file

FS Input Format Parameters

The FS input format supports the following parameters:

recurse

Values:	recursion level (number)
Default:	-1
Description:	Max subdirectory recursion level.
Details:	0 disables subdirectory recursion; -1 enables unlimited recursion.
Example:	-recurse:2

preserveLastAccTime

Values:	ON OFF
Default:	OFF
Description:	Preserve the last access time of visited files.
Details:	Enumerating files and directories causes their last access time to be updated. Setting this parameter to "ON" causes the FS input format to restore the last access time of the files being visited.
Example:	-preserveLastAccTime:ON

useLocalTime

Values:	ON OFF
Default:	ON
Description:	Use local time for timestamp fields.
Details:	When set to "ON", the values of the "CreationTime",

"LastAccessTime", and "LastWriteTime" fields are expressed in local time. When set to "OFF", the values of these fields are expressed in Universal Time Coordinates (UTC) time.

Example: -useLocalTime:OFF

FS Input Format Examples

Ten Largest Files

Print the 10 largest files on the C: drive:

LogParser "SELECT TOP 10 Path, Name, Size FROM C:*.* ORDER BY Siz e DESC" -i:FS

MD5 Hashes of System Files

Return the MD5 hash of system executable files:

```
LogParser "SELECT Path, HASHMD5_FILE(Path) FROM C:\Windows\Syste
m32\*.exe" -i:FS -recurse:0
```

Identical Files

Find out if there are identical copies of the same file on the C: drive:

```
LogParser "SELECT HASHMD5_FILE(Path) AS Hash, COUNT(*) AS Num
berOfCopies FROM C:\*.* GROUP BY Hash HAVING NumberOfCopies > 1
" -i:FS
```

HTTPERR Input Format

The HTTPERR input format parses HTTP Error log files created by the Http.sys driver.

HTTP Error log files are server-wide text log files containing log entries for Http.sys-initiated error responses to malformed client requests or to valid requests that are aborted due to abnormal circumstances.

Depending on the version of Http.sys, HTTP Error log files can be logged in two different formats.

Earlier versions of Http.sys log HTTP Error log entries as raw lines consisting of space-separated values. The following example shows a portion of an HTTP Error log file generated by earlier versions of Http.sys:

2002-06-27 19:11:28 172.30.92.88 3405 172.30.162.213 80 HTTP/1.0 GET /m

sadc/..%255c../..%255c../..%255c/..%c1%1c../..%c1%1c../..%c1%1c../.winnt/sy Laten yersions of / Http://woology/HTTP Error log files in the W3C Extended log/file/format9:1400/files/

The following example shows a portion of an HTTP Error log file generated by later versions of Http.sys:

#Software: Microsoft HTTP API 1.0

#Version: 1.0

#Date: 2003-08-08 03:12:41

Fields: date time c-ip c-port s-ip s-port cs-version cs-method cs-uri sc-status s

Parameters 03:12:41 10.193.50.9 3544 10.193.50.9 80 HTTP/1.1 GET /ISAPI COP/ISAPIExtTest.dll?Action=Crash&Action;=Print&Data;=Req17769_0 -1 Connection_Abandoned_By_AppPool DefaultAppPool

2003-08-08 03:12:41 10.193.50 9 3545 10.193 50 9 80 HTTP/1.1 GET /ISAPI © 2004 Microsoft Corporation. All rights reserved. _OOP/ISAPIExtTest.dll?Action=Crash&Action;=Print&Data;=Req17769_1 -1 Connection_Abandoned_By_AppPool DefaultAppPool 2003-08-08 03:12:43 10.193.50.9 3546 10.193.50.9 80 HTTP/1.1 GET /ISAPI _OOP/ISAPIExtTest.dll?Action=Crash&Action;=Print&Data;=Req17769_2 -1 Connection_Abandoned_By_AppPool DefaultAppPool

HTTPERR Input Format From-Entity Syntax

<from-entity> ::= HTTPERR |

<filename> [, <filename> ...]

The <<u>from-entity</u>> specified in queries using the HTTERR input format is either the "HTTPERR" keyword or a comma-separated list of paths of HTTP Error log files.

When the "HTTPERR" keyword is used, the HTTPERR input format reads the HTTP Error log configuration from the registry and parses all the HTTP Error log files currently available in the HTTP Error log file directory.

Filenames can include wildcards (e.g. "LogFiles\HTTPERR\httperr*.log").

Examples:

FROM LogFiles\HTTPERR\httperr1.log, LogFiles\HTTPERR\httperr2.log

FROM \\MYMACHINE\LogFiles\HTTPERR\httperr*.log

FROM HTTPERR

HTTPERR Input Format Fields

The input records generated by the HTTPERR input format contain the following fields:

Name	Туре	Description
LogFilename	STRING	Full path of the log file containing this entry
LogRow	INTEGER	Line in the log file containing this entry
date	TIMESTAMP	The date on which the request was served (Universal Time Coordinates (UTC) time)
time	TIMESTAMP	The time at which the request was served (Universal Time Coordinates (UTC) time)
s- computername	STRING	The name of the server that served the request (this field is logged by later versions of Http.sys only)
c-ip	STRING	The IP address of the client that made the request
c-port	INTEGER	The client port number that sent the request
s-ip	STRING	The IP address of the server that served the request

s-port	INTEGER	The server port number that received the request
cs-version	STRING	The HTTP version of the client request
cs-method	STRING	The HTTP request verb
cs-uri	STRING	The HTTP request uri
cs(User- Agent)	STRING	The client request User-Agent header (this field is logged by later versions of Http.sys only)
cs(Cookie)	STRING	The client request Cookie header (this field is logged by later versions of Http.sys only)
cs(Referer)	STRING	The client request Referer header (this field is logged by later versions of Http.sys only)
cs-host	STRING	The client request Host header (this field is logged by later versions of Http.sys only)
sc-status	INTEGER	The response HTTP status code
sc-bytes	INTEGER	The number of bytes in the response sent by the server (this field is logged by later versions of Http.sys only)
cs-bytes	INTEGER	The number of bytes in the request

		sent by the client (this field is logged by later versions of Http.sys only)
time-taken	INTEGER	The number of milliseconds elapsed since the moment the server received the request to the moment the server sent the response to the client (this field is logged by later versions of Http.sys only)
s-siteid	INTEGER	The IIS site instance number that served the request
s-reason	STRING	Information about why the error occurred
s-queuename	STRING	The name of the application pool hosting the IIS worker process that processed the request (this field is logged by later versions of Http.sys only)

HTTPERR Input Format Parameters

The HTTPERR input format supports the following parameters:

iCodepage

Values:	codepage ID (number)
Default:	0
Description:	Codepage of the log file.
Details:	0 is the system codepage, -1 is UNICODE.
Example:	-iCodepage:1245

minDateMod

Values:	date/time (in "yyyy-MM-dd hh:mm:ss" format)
Default:	not specified
Description:	Minimum file last modified date, in local time coordinates.
Details:	When this parameter is specified, the HTTPERR input format processes only log files that have been modified after the specified date.
Example:	-minDateMod:"2004-05-28 22:05:10"

dirTime

Values:	ON	
, and co.	••••	••••

Default: OFF

Description: Use the value of the "#Date" directive for the "date" and/or "time" field values when these fields are not logged.

Details:	When a log file is configured to not log the "date" and/or
	"time" fields, specifying "ON" for this parameters causes
	the HTTPERR input format to generate "date" and
	"time" values using the value of the last seen "#Date"
	directive.

Example: -dirTime:ON

iCheckpoint

Values:	checkpoint filename
---------	---------------------

Default: *not specified*

Description: Load and save checkpoint information to this file.

Details: This parameter enables the "Incremental Parsing" feature that allows sequential executions of the same query to only process new log entries that have been logged since the last execution. For more information, see <u>Parsing Input Incrementally</u>.

Example: -iCheckpoint:C:\Temp\myCheckpoint.lpc

HTTPERR Input Format Examples

Errors Distribution Chart

Create a pie chart containing the distribution of errors in the HTTP Error logs:

LogParser "SELECT sc-status, PROPCOUNT(*) AS Percentage INTO Pie.gif FROM HTTPERR GROUP BY sc-status ORDER BY Percentage DESC" -cha rtType:PieExploded -chartTitle:"Errors Distribution" -categories:off

IIS Input Format

The IIS input format parses IIS log files in the Microsoft IIS Log File Format.

The Microsoft IIS Log File Format is a text-based, fixed-field format. Log entries are logged on a single line, consisting of a comma-separated list of field values.

The following example shows a portion of a Microsoft IIS Log File Format log file:

192.168.114.201, -, 03/20/01, 7:55:20, W3SVC2, SERVER, 172.21.13.45, 450 2, 163, 3223, 200, 0, GET, /DeptLogo.gif, -, 192.168.110.54, -, 03/20/01, 7:57:20, W3SVC2, SERVER, 172.21.13.45, 411, 221, 1967, 200, 0, GET, /style.css, -, Fields Parameters Examples

See also:

IIS Output Format

IIS Input Format From-Entity Syntax

<from-entity></from-entity>	::=	<filename> <siteid> [, <filename> <siteid>]</siteid></filename></siteid></filename>	
<siteid></siteid>	::=	< site_number > < server_comment > < site_metabase_path >	

The <<u>from-entity</u>> specified in queries using the IIS input format is a comma-separated list of:

- Paths of Microsoft IIS Log File Format log files;
- IIS Virtual Site "identifiers".

"Site identifiers" must be enclosed within angle brackets (< and >), and can have one of the following values:

- The numeric site ID (e.g. "<1>", "<28163489>");
- The text value of the "ServerComment" property of the site (e.g. "<My External Site>", "<www.margiestravel.com>");
- The fully-qualified ADSI metabase path to the site (e.g. " <//MYSERVER/W3SVC/1>"), using either the numeric site ID or the text value of the "ServerComment" property of the site.

When a "site identifier" is used, the IIS input format connects to the specified machine's metabase, gathers information on the site's current logging properties, and parses all the log files in the site's current log file directory.

Filenames and "Site identifiers" can also include wildcards (e.g. "LogFiles\in04*.log", "<www.*.com>").

Examples:

FROM LogFiles\in04*log, LogFiles\in03*.log, \\MyServer\LoggingShare\W3 SVC2\in04*.log

FROM <1>, <2>, <My External Site>, inetsv9.log

FROM <www.net*home.com>, <//MyServer2/W3SVC/www.net*home.com>, <//MyServer2/MSFTPSVC/*>, <*>

IIS Input Format Fields

The input records generated by the IIS input format contain the following fields:

Name	Туре	Description
LogFilename	STRING	Full path of the log file containing this entry
LogRow	INTEGER	Line in the log file containing this entry
UserIP	STRING	The IP address of the client that made the request
UserName	STRING	The name of the authenticated user that made the request, or NULL if the request was from an anonymous user
Date	TIMESTAMP	The date on which the request was served (local time)
Time	TIMESTAMP	The time at which the request was served (local time)
ServiceInstance	STRING	The IIS service name and site instance number that served the request
HostName	STRING	The name of the server that served the request
ServerIP	STRING	The IP address of the server that served the request
-----------------	---------	--
TimeTaken	INTEGER	The number of milliseconds elapsed since the moment the server received the request to the moment the server sent the last response chunk to the client
BytesSent	INTEGER	The number of bytes in the request sent by the client
BytesReceived	INTEGER	The number of bytes in the response sent by the server
StatusCode	INTEGER	The response HTTP or FTP status code
Win32StatusCode	INTEGER	The Windows status code associated with the response HTTP or FTP status code
RequestType	STRING	The HTTP request verb or FTP operation
Target	STRING	The HTTP request uri-stem or FTP operation target
Parameters	STRING	The HTTP request uri-query, or NULL if the requested URI did not include a uri-query

IIS Input Format Parameters

The IIS input format supports the following parameters:

iCodepage

Values:	codepage ID (number)
Default:	-2
Description:	Codepage of the log file.
Details:	0 is the system codepage; -2 specifies that the codepage is automatically determined by inspecting the filename and/or the site's "LogInUTF8" property.
Example:	-iCodepage:1245

recurse

Values:	recursion level (number)
Default:	0
Description:	Max subdirectory recursion level.
Details:	0 disables subdirectory recursion; -1 enables unlimited recursion.
Example:	-recurse:-1

minDateMod

Values:	date/time (in "yyyy-MM-dd hh:mm:ss" format)
Default:	not specified
Description:	Minimum file last modified date, in local time coordinates.
Details:	When this parameter is specified, the IIS input format

processes only log files that have been modified after the specified date.

Example: -minDateMod:"2004-05-28 22:05:10"

locale

- Values: 3-character locale ID
- Default: DEF

Description: ID of the locale in which the log file was generated.

Details: IIS versions earlier than 6.0 log the "Date" and "Time" fields using the current system locale date and time formats. IIS 6.0 and later versions use the ENU locale instead, regardless of the system locale settings. For these reasons, when parsing Microsoft IIS Log File Format log files on a locale whose date and time formats do not match the formats of the locale of the computer where the log file has been created, users need to specify the ID of the system locale of the computer that created the log file. The special "DEF" value means the current system locale.

Example:

-locale:JPN

iCheckpoint

Values: checkpoint filename
Default: not specified
Description: Load and save checkpoint information to this file.
Details: This parameter enables the "Incremental Parsing" feature that allows sequential executions of the same query to only process new log entries that have been logged since the last execution. For more information, see Parsing Input Incrementally.

Example: -iCheckpoint:C:\Temp\myCheckpoint.lpc

IIS Input Format Examples

Top 20 URL's for a Site

Create a chart containing the TOP 20 URL's in the "www.margiestravel.com" web site (assumed to be logging in the IIS log format):

LogParser "SELECT TOP 20 Target, COUNT(*) AS Hits INTO MyChart.gif FROM <www.margiestravel.com> GROUP BY Target ORDER BY Hits DES C" -chartType:Column3D -groupSize:1024x768

Export Errors to SYSLOG

Send error entries in the IIS log to a SYSLOG server:

LogParser "SELECT TO_TIMESTAMP(Date, Time), CASE StatusCode WHE N 500 THEN 'emerg' ELSE 'err' END AS MySeverity, HostName AS MyHost name, Target INTO @myserver FROM <1> WHERE StatusCode >= 400" -o: SYSLOG -severity:\$MySeverity -hostName:\$MyHostname Bytes by Extension Chart

Bytes by Extension Chart

Create a pie chart with the total number of bytes generated by each extension:

LogParser "SELECT EXTRACT_EXTENSION(Target) AS Extension, MUL(PROPSUM(BytesReceived),100.0) AS Bytes INTO Pie.gif FROM <1> GRO UP BY Extension ORDER BY Bytes DESC" -chartType:PieExploded -chartTi -tle:"Bytes per extension" -categories:off

IISODBC Input Format

The IISODBC input format returns database records from the tables logged to by IIS when configured to log in the ODBC Log Format.

From-Entity Syntax Fields Examples

IISODBC Input Format From-Entity Syntax

<from-< th=""><th>::=</th><th><siteid> [, <siteid>] </siteid></siteid></th></from-<>	::=	<siteid> [, <siteid>] </siteid></siteid>
entity>		table: <tablename>;username:<username>;password:</username></tablename>
		<password>;dsn:<dsn></dsn></password>
<siteid></siteid>	::=	< site_number > < server comment >
		< site_metabase_path >

The <<u>from-entity</u>> specified in queries using the IISODBC input format is either a comma-separated list of IIS Virtual Site "identifiers", or a single specification of the ODBC parameters needed to access the table.

"Site identifiers" must be enclosed within angle brackets (< and >), and can have one of the following values:

- The numeric site ID (e.g. "<1>", "<28163489>");
- The text value of the "ServerComment" property of the site (e.g. "<My External Site>", "<www.margiestravel.com>");
- The fully-qualified ADSI metabase path to the site (e.g. " <//MYSERVER/W3SVC/1>"), using either the numeric site ID or the text value of the "ServerComment" property of the site.

When a "site identifier" is used, the IISODBC input format connects to the specified machine's metabase, gathers information on the site's current ODBC logging properties, and uses this information to connect to the database table.

"Site identifiers" can also include wildcards (e.g. "<www.*.com>").

Examples:

FROM <1>, <2>, <My External Site>

FROM table:MYLOGTABLE;username:IISLOGUSER;password:IISLOGUS ERPW;dsn:IISLOGDSN

IISODBC Input Format Fields

The input records generated by the IISODBC input format contain the following fields:

Name	Туре	Description
ClientHost	STRING	The IP address of the client that made the request
Username	STRING	The name of the authenticated user that made the request, or NULL if the request was from an anonymous user
LogTime	TIMESTAMP	The date and time at which the request was served (local time)
Service	INTEGER	The IIS service name and site instance number that served the request
Machine	STRING	The name of the server that served the request
ServerIP	STRING	The IP address of the server that served the request
ProcessingTime	INTEGER	The number of milliseconds elapsed since the moment the server received the request to the moment the server sent the last

		response chunk to the client
BytesRecvd	INTEGER	The number of bytes in the request sent by the client
BytesSent	INTEGER	The number of bytes in the response sent by the server
ServiceStatus	INTEGER	The response HTTP or FTP status code
Win32Status	INTEGER	The Windows status code associated with the response HTTP or FTP status code
Operation	STRING	The HTTP request verb or FTP operation
Target	STRING	The HTTP request uri-stem or FTP operation target
Parameters	STRING	The HTTP request uri-query, or NULL if the requested URI did not include a uri-query

IISODBC Input Format Examples

Top 20 URL's for a Site

Create a chart containing the TOP 20 URL's in the "www.margiestravel.com" web site (assumed to be logging in the ODBC log format):

LogParser "SELECT TOP 20 Target, COUNT(*) AS Hits INTO MyChart.gif FROM <www.margiestravel.com> GROUP BY Target ORDER BY Hits DES C" -chartType:Column3D -groupSize:1024x768

IISW3C Input Format

The IISW3C input format parses IIS log files in the W3C Extended Log File Format.

IIS web sites logging in the W3C Extended format can be configured to log only a specific subset of the available fields.

Log files in this format begin with some informative headers ("directives"), the most important of which is the "#Fields" directive, describing which fields are logged at which position in a log row.

After the directives, the log entries follow. Each log entry is a spaceseparated list of field values.

If the logging configuration of an IIS virtual site is updated, the structure of the fields in the file that is currently logged to might change according to the new configuration. In this case, a new "#Fields" directive is logged describing the new fields structure, and the IISW3C input format keeps track of the structure change and parses the new log entries accordingly.

The following example shows a portion of a W3C Extended Log File Format log file:

See also:

W3C Input Format W3C Output Format

IISW3C Input Format From-Entity Syntax

<from-entity></from-entity>	::=	<filename> <siteid> [, <filename> <siteid>]</siteid></filename></siteid></filename>
<siteid></siteid>	::=	< site_number > < server_comment >
		< site_metabase_path >

The <from-entity> specified in queries using the IISW3C input format is a comma-separated list of:

- Paths of IIS W3C Extended log files;
- IIS Virtual Site "identifiers".

"Site identifiers" must be enclosed within angle brackets (< and >), and can have one of the following values:

- The numeric site ID (e.g. "<1>", "<28163489>");
- The text value of the "ServerComment" property of the site (e.g. "<My External Site>", "<www.margiestravel.com>");
- The fully-qualified ADSI metabase path to the site (e.g. " <//MYSERVER/W3SVC/1>"), using either the numeric site ID or the text value of the "ServerComment" property of the site.

When a "site identifier" is used, the IISW3C input format connects to the specified machine's metabase, gathers information on the site's current logging properties, and parses all the log files in the site's current log file directory.

Filenames and "Site identifiers" can also include wildcards (e.g. "LogFiles\ex04*.log", "<www.*.com>").

Examples:

FROM LogFiles\ex04*log, LogFiles\ex03*.log, \\MyServer\LoggingShare\W3 SVC2\ex04*.log

FROM <1>, <2>, <My External Site>, extend9.log

FROM <www.net*home.com>, <//MyServer2/W3SVC/www.net*home.com>, <//MyServer2/MSFTPSVC/*>, <*>

IISW3C Input Format Fields

The input records generated by the IISW3C input format contain the following fields:

Name	Туре	Description
LogFilename	STRING	Full path of the log file containing this entry
LogRow	INTEGER	Line in the log file containing this entry
date	TIMESTAMP	The date on which the request was served (Universal Time Coordinates (UTC) time)
time	TIMESTAMP	The time at which the request was served (Universal Time Coordinates (UTC) time)
c-ip	STRING	The IP address of the client that made the request
cs-username	STRING	The name of the authenticated user that made the request, or NULL if the request was from an anonymous user
s-sitename	STRING	The IIS service name and site instance number that served the request

s- computername	STRING	The name of the server that served the request
s-ip	STRING	The IP address of the server that served the request
s-port	INTEGER	The server port number that received the request
cs-method	STRING	The HTTP request verb or FTP operation
cs-uri-stem	STRING	The HTTP request uri-stem or FTP operation target
cs-uri-query	STRING	The HTTP request uri-query, or NULL if the requested URI did not include a uri-query
sc-status	INTEGER	The response HTTP or FTP status code
sc-substatus	INTEGER	The response HTTP sub-status code (this field is logged by IIS version 6.0 and later only)
sc-win32- status	INTEGER	The Windows status code associated with the response HTTP or FTP status code
sc-bytes	INTEGER	The number of bytes in the response sent by the server
cs-bytes	INTEGER	The number of bytes in the request

		sent by the client
time-taken	INTEGER	The number of milliseconds elapsed since the moment the server received the request to the moment the server sent the last response chunk to the client
cs-version	STRING	The HTTP version of the client request
cs-host	STRING	The client request Host header
cs(User- Agent)	STRING	The client request User-Agent header
cs(Cookie)	STRING	The client request Cookie header
cs(Referer)	STRING	The client request Referer header
s-event	STRING	The type of log event (this field is logged by IIS version 5.0 only when the "Process Accounting Logging" feature is enabled)
s-process-type	STRING	The type of process that triggered the log event (this field is logged by IIS version 5.0 only when the "Process Accounting Logging" feature is enabled)
s-user-time	REAL	The total accumulated User Mode processor time, in percentage, that

		the site used during the current interval (this field is logged by IIS version 5.0 only when the "Process Accounting Logging" feature is enabled)
s-kernel-time	REAL	The total accumulated Kernel Mode processor time, in percentage, that the site used during the current interval (this field is logged by IIS version 5.0 only when the "Process Accounting Logging" feature is enabled)
s-page-faults	INTEGER	The total number of memory references that resulted in memory page faults during the current interval (this field is logged by IIS version 5.0 only when the "Process Accounting Logging" feature is enabled)
s-total-procs	INTEGER	The total number of applications created during the current interval (this field is logged by IIS version 5.0 only when the "Process Accounting Logging" feature is enabled)
s-active-procs	INTEGER	The total number of applications

		running when the log event was triggered (this field is logged by IIS version 5.0 only when the "Process Accounting Logging" feature is enabled)
s-stopped- procs	INTEGER	The total number of applications stopped due to process throttling during the current interval <i>(this field is logged by IIS version 5.0 only when the "Process Accounting Logging" feature is enabled)</i>

IISW3C Input Format Parameters

The IISW3C input format supports the following parameters:

iCodepage

Values:	codepage ID (number)
Default:	-2
Description:	Codepage of the log file.
Details:	0 is the system codepage; -2 specifies that the codepage is automatically determined by inspecting the filename and/or the site's "LogInUTF8" property.
Example:	-iCodepage:1245

recurse

Values:	recursion level (number)
Default:	0
Description:	Max subdirectory recursion level.
Details:	0 disables subdirectory recursion; -1 enables unlimited recursion.
Example:	-recurse:-1

minDateMod

Values:	date/time (in "yyyy-MM-dd hh:mm:ss" format)
Default:	not specified
Description:	Minimum file last modified date, in local time coordinates.
Details:	When this parameter is specified, the IISW3C input

format processes only log files that have been modified after the specified date.

Example:	-minDateMod:"2004-05-28 22:05:10"
----------	-----------------------------------

dQuotes

Values:	ON OFF
Default:	OFF
Description:	Specifies that string values in the log are double- quoted.
Details:	Log processors might generate W3C logs whose string values are enclosed in double-quotes.
Example:	-dQuotes:ON

dirTime

Values:	ON	OFF
Values:	ON	

Default: OFF

Description: Use the value of the "#Date" directive for the "date" and/or "time" field values when these fields are not logged.

Details: When a log file is configured to not log the "date" and/or "time" fields, specifying "ON" for this parameters causes the IISW3C input format to generate "date" and "time" values using the value of the last seen "#Date" directive.

Example: -dirTime:ON

consolidateLogs

Values: **ON | OFF**

Default: OFF

- Description: Return entries from all the input log files ordering by date and time.
- Details: When a from-entity refers to log files from multiple IIS virtual sites, specifying ON for this parameter causes the IISW3C input format to parse all the input log files in parallel, returning entries ordered by the values of the "date" and "time" fields in the log files; the input records returned will thus appear as if a single IISW3C log file was being parsed.
 Enabling this feature is equivalent to executing a query with an "ORDER BY date, time" clause on all the log files. However, the implementation of this feature leverages the pre-existing chronological order of entries in each log file, and it does not require the extensive memory resources otherwise required by the ORDER

Example: -consolidateLogs:ON

iCheckpoint

Values: checkpoint filename

BY query clause.

Default: *not specified*

Description: Load and save checkpoint information to this file.

Details: This parameter enables the "Incremental Parsing" feature that allows sequential executions of the same query to only process new log entries that have been logged since the last execution. For more information, see <u>Parsing Input Incrementally</u>.

Example: -iCheckpoint:C:\Temp\myCheckpoint.lpc

IISW3C Input Format Examples

Top 20 URL's for a Site

Create a chart containing the TOP 20 URL's in the "www.margiestravel.com" web site (assumed to be logging in the W3C log format):

LogParser "SELECT TOP 20 cs-uri-stem, COUNT(*) AS Hits INTO MyChart .gif FROM <www.margiestravel.com> GROUP BY cs-uri-stem ORDER BY Hits DESC" -chartType:Column3D -groupSize:1024x768

Export Errors to SYSLOG

Send error entries in the W3C log to a SYSLOG server:

LogParser "SELECT TO_TIMESTAMP(date,time), CASE sc-status WHEN 5 00 THEN 'emerg' ELSE 'err' END AS MySeverity, s-computername AS MyHo stname, cs-uri-stem INTO @myserver FROM <1> WHERE sc-status >= 400" -o:SYSLOG -severity:\$MySeverity -hostName:\$MyHostname Bytes by Extension Chart

Create a pie chart with the total number of bytes generated by each extension:

LogParser "SELECT EXTRACT_EXTENSION(cs-uri-stem) AS Extension, M UL(PROPSUM(sc-bytes),100.0) AS Bytes INTO Pie.gif FROM <1> GROUP BY Extension ORDER BY Bytes DESC" -chartType:PieExploded -chartTitle:" _Bytes per extension" -categories:off

NCSA Input Format

The NCSA input format parses log files in the NCSA Common, Combined, and Extended Log File Formats.

The NCSA Log File Format is a text-based, fixed-field format. Log entries are logged on a single line, consisting of a space-separated list of field values.

There are three versions of the NCSA Log File Format: "Common", "Combined", and "Extended". The three versions differ by the number of fields that are logged for each request.

IIS can log NCSA Common Log File Format log files, while other web servers can be configured to log with the Combined and Extended formats.

The following example shows a portion of an NCSA Common Log File Format log file:

172.21.13.45 - Microsoft\User [08/Apr/2001:17:39:04 -0800] "GET /scripts/iis admin/ism.dll?http/serv HTTP/1.0" 200 3401

The following example is the formation of a model of the formation of the formation of a model of the formation of a model of the formation of a model of the formation of the formation of a model of the formation of the format

172.21.13.45 - Microsoft\User [08/Apr/2001:17:39:04 -0800] "GET /scripts/iis admin/ism.dll?http/serv HTTP/1.0" 200 3401 "http://www.microsoft.com/" "M ozilla/4.05 [en] (WinNT; I)" "USERID=CustomerA"

From-Entity Syntax [08/Apr/2001:21:01:19 -0800] "GET /style.css HTTP/1.0" From-Entity Syntax [08/Apr/2001:21:01:19 -0800] "GET /style.css HTTP/1.0" From 1937 "http://www.microsoft.com/" "Mozilla/4.05 [en] (WinNT; I)" "USER Parameters"

Examples

NCSA Input Format From-Entity Syntax

<from-entity></from-entity>	::=	<filename> <siteid> [, <filename> <siteid>]</siteid></filename></siteid></filename>
<siteid></siteid>	::=	< site_number > < server_comment > < site_metabase_path >

The <<u>from-entity</u>> specified in queries using the NCSA input format is a comma-separated list of:

- Paths of NCSA Log File Format log files;
- IIS Virtual Site "identifiers".

"Site identifiers" must be enclosed within angle brackets (< and >), and can have one of the following values:

- The numeric site ID (e.g. "<1>", "<28163489>");
- The text value of the "ServerComment" property of the site (e.g. "<My External Site>", "<www.margiestravel.com>");
- The fully-qualified ADSI metabase path to the site (e.g. " <//MYSERVER/W3SVC/1>"), using either the numeric site ID or the text value of the "ServerComment" property of the site.

When a "site identifier" is used, the NCSA input format connects to the specified machine's metabase, gathers information on the site's current logging properties, and parses all the log files in the site's current log file directory.

Filenames and "Site identifiers" can also include wildcards (e.g. "LogFiles\nc04*.log", "<www.*.com>").

Examples:

FROM LogFiles\nc04*log, LogFiles\nc03*.log, \\MyServer\LoggingShare\W3 SVC2\nc04*.log

FROM <1>, <2>, <My External Site>, ncsa9.log

FROM <www.net*home.com>, <//MyServer2/W3SVC/www.net*home.com>, </*>

NCSA Input Format Fields

The input records generated by the NCSA input format contain the following fields:

Name	Туре	Description
LogFilename	STRING	Full path of the log file containing this entry
LogRow	INTEGER	Line in the log file containing this entry
RemoteHostName	STRING	The IP address of the client that made the request
RemoteLogName	STRING	The identifier used to identify the client making the HTTP request, or NULL if no identifier is used (always NULL in NCSA log files generated by IIS)
UserName	STRING	The name of the authenticated user that made the request, or NULL if the request was from an anonymous user
DateTime	TIMESTAMP	The date and time at which the request was served (Universal Time Coordinates (UTC) time)
Request	STRING	The HTTP request line (verb,

		URI, and HTTP version)
StatusCode	INTEGER	The response HTTP status code
BytesSent	INTEGER	The number of bytes in the response sent by the server
Referer	STRING	The client request Referer header (not logged in NCSA Common Log File Format log files)
User-Agent	STRING	The client request User-Agent header (not logged in NCSA Common Log File Format log files)
Cookie	STRING	The client request Cookie header (not logged in NCSA Common Log File Format log files)

NCSA Input Format Parameters

The NCSA input format supports the following parameters:

iCodepage

Values:	codepage ID (number)
Default:	-2
Description:	Codepage of the log file.
Details:	0 is the system codepage; -2 specifies that the codepage is automatically determined by inspecting the filename and/or the site's "LogInUTF8" property.
Example:	-iCodepage:1245

recurse

Values:	recursion level (number)
Default:	0
Description:	Max subdirectory recursion level.
Details:	0 disables subdirectory recursion; -1 enables unlimited recursion.
Example:	-recurse:-1

minDateMod

Values:	date/time (in "yyyy-MM-dd hh:mm:ss" format)
Default:	not specified
Description:	Minimum file last modified date, in local time coordinates.
Details:	When this parameter is specified, the NCSA input

format processes only log files that have been modified after the specified date.

Example: -minDateMod:"2004-05-28 22:05:10"

iCheckpoint

Values:	checkpoint filename	
Default:	not specified	
Description:	Load and save checkpoint information to this file.	
Details:	This parameter enables the "Incremental Parsing" feature that allows sequential executions of the same query to only process new log entries that have been logged since the last execution. For more information, see <u>Parsing Input Incrementally</u> .	
Example:	-iCheckpoint:C:\Temp\myCheckpoint.lpc	

NCSA Input Format Examples

Slice Request field into components

Return the verb, URI, and HTTP version for each request:

LogParser "SELECT EXTRACT_TOKEN(Request, 0, ' ') AS Verb, EXTRAC T_TOKEN(Request, 1, ' ') AS URI, EXTRACT_TOKEN(Request, 2, ' ') AS V ersion FROM ncsa9.log"

Top 20 URL's for a Site

Create a chart containing the TOP 20 URL's in the "www.margiestravel.com" web site (assumed to be logging in the NCSA log format):

LogParser "SELECT TOP 20 EXTRACT_TOKEN(Request, 1, ' ') AS URI, C OUNT(*) AS Hits INTO MyChart.gif FROM <www.margiestravel.com> GR OUP BY URI ORDER BY Hits DESC" -chartType:Column3D -groupSize:102 _4x768

NETMON Input Format

The NETMON input format parses network capture files (.cap files) created by the NetMon Network Monitor application.

The NETMON input format works in two different modes, selectable through the \underline{fMode} parameter.

When the "fMode" parameter is set to "TCPIP", the NETMON input format returns an input record for each TCP/IP packet found in the capture file.

In this case, input records contain fields from the TCP and IP packet headers, together with the payload of each packet.

For example, the following command returns the specified fields from the TCP/IP packets in the capture file:

LogParser "SELECT SrcPort, TCPFlags, PayloadBytes FROM MyCapture.cap " -fMode:TCPIP

The output of this command would look like the following sample:

SrcPort TCPFlags PayloadBytes

Wagen the "fMpde" parameter is set to "TCPConn", the NETMON input format raturnsoan input record for each TCP connection found in the capsure file. 0

Ingohis case, imput records contain fields calculated by aggregating all the TCB packets in the connection, including the reconstructed payload sent by south and points.

For several provides the specified fields from the T&P connections in the capture file:

1336 A 1431

Lagearsep"SE549CT SrcPort, TimeTaken, SrcPayloadBytes, DstPayloadBytes

FROM MyCapture.cap" -fMode:TCPConn

The output of this command would look like the following sample:

SrcPort TimeTaken SrcPayloadBytes DstPayloadBytes

----- ------

1336	150.216000 3694	3673	
_1284	450.648000 312	1362	
<u>-1286</u>	<u>-ntity Syntax</u> 711.023000 0	0	
<u>Fields</u>	1001.440000 0	0	
<u>Palalin</u> _1288	efers 851.224000 0	0	
Examp 1289	les 15120.240000 0	0	
1283	66619.388000 1886	3718	
1291	1366 3.102000 312	ft Corperation. All rights reserved.	
1285	47883.357000 312	708	
1290	21203.946000 312	1362	

NETMON Input Format From-Entity Syntax

<from-entity> ::= <filename> [, <filename> ...]

The <from-entity> specified in queries using the NETMON input format is a comma-separated list of NetMon capture files (.cap files).

Examples:

FROM MyCapture1.cap

FROM MyCapture1.cap, MyCapture2.cap
NETMON Input Format Fields

The structure of the input records generated by the NETMON input format depends on the value specified for the <u>fMode</u> parameter.

TCPIP Mode

When the <u>fMode</u> parameter is set to "TCPIP", the NETMON input format returns an input record for each TCP/IP packet found in the capture file. In this mode, input records contain the following fields:

Name	Туре	Description
CaptureFilename	STRING	The full path of the capture file containing this packet
Frame	INTEGER	The frame number containing this packet
DateTime	TIMESTAMP	Date and time at which the packet was sent
FrameBytes	INTEGER	Total number of bytes in the frame
SrcMAC	STRING	MAC address of the sender of this packet
SrcIP	STRING	IP address of the sender of this packet
SrcPort	INTEGER	TCP port number of the sender of this packet

DstMAC	STRING	MAC address of the destination of this packet
DstIP	STRING	IP address of the destination of this packet
DstPort	INTEGER	TCP port number of the destination of this packet
IPVersion	INTEGER	IP version of this packet
TTL	INTEGER	Time-To-Live field of the IP header of this packet
TCPFlags	STRING	TCP flags field of the TCP header of this packet
Seq	INTEGER	TCP sequence number of this packet
Ack	INTEGER	TCP acknowledge number of this packet
WindowSize	INTEGER	Window size field of the TCP header of this packet
PayloadBytes	INTEGER	Number of bytes in the TCP payload of this packet
Payload	STRING	TCP payload of this packet
Connection	INTEGER	Unique identifier of the TCP connection to which this packet belongs

TCPConn Mode

When the <u>fMode</u> parameter is set to "TCPConn", the NETMON input format returns an input record for each TCP connection found in the capture file.

In this mode, input records contain the following fields:

Name	Туре	Description
CaptureFilename	STRING	The full path of the capture file containing this connection
StartFrame	INTEGER	Frame number containing the first packet of this connection
EndFrame	INTEGER	Frame number containing the last packet of this connection
Frames	INTEGER	Total number of frames containing packets belonging to this connection
DateTime	TIMESTAMP	Date and time of at which the first packet of this connection was sent
TimeTaken	INTEGER	Total number of milliseconds elapsed since the first packet of this connection to the last packet
SrcMAC	STRING	MAC address of the initiator of this connection

	1	III
SrcIP	STRING	IP address of the initiator of this connection
SrcPort	INTEGER	TCP port number of the initiator of this connection
SrcPayloadBytes	INTEGER	Total number of bytes in the reconstructed TCP payload sent by the initiator of this connection
SrcPayload	STRING	Reconstructed TCP payload sent by the initiator of this connection
DstMAC	STRING	MAC address of the receiver of this connection
DstIP	STRING	IP address of the receiver of this connection
DstPort	INTEGER	TCP port number of the receiver of this connection
DstPayloadBytes	INTEGER	Total number of bytes in the reconstructed TCP payload sent by the receiver of this connection
DstPayload	STRING	Reconstructed TCP payload sent by the receiver of this connection

NETMON Input Format Parameters

The NETMON input format supports the following parameters:

fMode

Values:	TCPIP TCPConn
Default:	TCPIP
Description:	Operation mode.
Details:	When this parameter is set to "TCPIP", the NETMON increased for each TCP/IP packet found in the capture file. I contain fields from the TCP and IP packet headers, toge each packet. When this parameter is set to "TCPConn", the NETMON input record for each TCP connection found in the captur records contain fields calculated by aggregating all the T connection, including the reconstructed payload sent by For more information on the different modes of operation Format Fields.

Example: -fMode:TCPConn

binaryFormat

- Values: ASC | PRINT | HEX
- Default: ASC

Description: Format of binary fields.

Details: TCP packet payloads are returned as STRING values fo value specified for this parameter. When this parameter is set to "ASC", data bytes belongin are returned as ASCII characters, while data bytes outsin as period (.) characters, as shown in the following examp POST /test_system/request HTTP/1.1..Content-Length: 3411

When this parameter is set to "PRINT", data bytes reprecharacters are returned as ASCII characters, while data printable ASCII characters are returned as period (.) cha following example:

POST /test_system/request HTTP/1.1

Content-Length: 3411 Whom this parameter is set to "HEX", all data bytes are r hexadecimal values, as shown in the following example:

504F5354202F63636D5F73797374656D2F72657175657374

Example: -binaryFormat:PRINT

NETMON Input Format Examples

Network Traffic per Second

Display total network traffic bytes per second:

LogParser "SELECT QUANTIZE(DateTime, 1) AS Second, SUM(FrameByte s) INTO DATAGRID FROM MyCapture.cap GROUP BY Second"

REG Input Format

The REG input format returns information on registry values.

The REG input format enumerates local or remote registry keys and values, returning an input record for each registry value found in the enumeration.

From-Entity Syntax Fields Parameters Examples

See also:

FS Input Format

REG Input Format From-Entity Syntax

<from-entity></from-entity>	::=	<registry_key> [, <registry_key>]</registry_key></registry_key>
<registry_key></registry_key>	::=	[\\ <computer_name>]\[<root_name>[\ <subkey_path>]]</subkey_path></root_name></computer_name>
<root_name></root_name>	::=	HKCR HKCU HKLM HKCC HKU

The <from-entity> specified in queries using the REG input format is a comma-separated list of registry keys. Valid registry keys are:

- The registry root (e.g. "\");
- A system registry root (e.g. "\HKLM");
- Any key below a system registry root (e.g. "\HKLM\Software\Microsoft").

Registry keys can be optionally preceded by a remote computer name in the UNC notation.

Examples:

 $FROM \setminus$

FROM \HKLM, \HKCU

FROM \\SERVER1\HKLM\Software, \\SERVER2\HKLM\Software

REG Input Format Fields

The input records generated by the REG input format contain the following fields:

Name	Туре	Description
ComputerName	STRING	Name of the computer hosting the registry containing this value
Path	STRING	Path of the registry key containing this value
KeyName	STRING	Name of the registry key containing this value
ValueName	STRING	Name of the registry value
ValueType	STRING	Name of the type of the registry value
Value	STRING	Text representation of the content of the registry value
LastWriteTime	TIMESTAMP	Date and time at which the registry value has been last modified (Universal Time Coordinates (UTC) time)

REG Input Format Parameters

The REG input format supports the following parameters:

recurse

Values:	recursion level (number)
Default:	-1
Description:	Max subkey recursion level.
Details:	0 disables subkey recursion; -1 enables unlimited recurs
Example:	-recurse:2

multiSZSep

ny string

I

Default:

Description: Separator between elements of MULTI_SZ registry value

Details: Registry values of the MULTI_SZ type contain *arrays* of : cases, the content of the "Value" field is built by concater elements one after the other, using the value of this para separator between the elements.

Example: -multiSZSep:,

binaryFormat

Values:	ASC PRINT HEX
Default:	ASC
Description:	Format of REG_BINARY registry values.
Details:	Registry values of the REG_BINARY type contain binary often not suitable to be textually represented. This paran

how binary data is formatted to a STRING when returned the "Value" field.

When this parameter is set to "ASC", data bytes belongii 0x7F range are returned as ASCII characters, while data the range are returned as period (.) characters, as shown example:

Bucket: 02096553..rundll32.exe

When this parameter is set to "PRINT", data bytes repres ASCII characters are returned as ASCII characters, while do not represent printable ASCII characters are returned characters, as shown in the following example:

Bucket: 02096553

rundll32.exe

When this parameter is set to "HEX", all data bytes are redigit hexadecimal values, as shown in the following exam

4275636B65743A2030323039363535330D0A72756E646C(

Example:

-binaryFormat:PRINT

REG Input Format Examples

Upload Registry to SQL Table

Load a portion of the registry into a SQL table:

LogParser "SELECT * INTO MyTable FROM \HKLM" -i:REG -o:SQL -serve r:MyServer -database:MyDatabase -driver:"SQL Server" -username:TestSQLU ser -password:TestSQLPassword -createTable:ON

Registry Type Distribution

Display the distribution of registry value types:

LogParser "SELECT ValueType, COUNT(*) INTO DATAGRID FROM \HKL M GROUP BY ValueType"

TEXTLINE Input Format

The TEXTLINE input format returns lines from generic text files.

The TEXTLINE input format makes it possible to parse text files in any format not supported natively by Log Parser, and retrieve entire lines of text as a single field.

The field can then be processed by the SQL-like query by making use of string manipulation functions, such as the <u>EXTRACT_TOKEN</u> function.

From-Entity Syntax Fields Parameters Examples

See also:

TEXTWORD Input Format TSV Input Format

TEXTLINE Input Format From-Entity Syntax

<from-entity> ::= <filename> [, <filename> ...] | http://<url> | STDIN

The <from-entity> specified in queries using the TEXTLINE input format is either:

- A comma-separated list of paths to text files, eventually including wildcards;
- The URL of a text file;
- The "STDIN" keyword, which specifies that the input data is available from the input stream (commonly used when piping command executions).

Examples:

FROM *.txt, \\MyServer\FileShare*.tsv

FROM http://www.microsoft.adatum.com/example.tsv

type data.txt | LogParser "SELECT * FROM STDIN" -i:TEXTLINE

TEXTLINE Input Format Fields

The input records generated by the TEXTLINE input format contain the following fields:

Name	Туре	Description
LogFilename	STRING	Full path of the file containing this line
Index	INTEGER	Line number
Text	STRING	Text line content

TEXTLINE Input Format Parameters

The TEXTLINE input format supports the following parameters:

iCodepage

Values:	codepage ID (number)
Default:	0
Description:	Codepage of the text file.
Details:	0 is the system codepage, -1 is UNICODE.
Example:	-iCodepage:1245

recurse

Values:	recursion level (number)
Default:	0
Description:	Max subdirectory recursion level.
Details:	0 disables subdirectory recursion; -1 enables unlimited recursion.
Example:	-recurse:-1

splitLongLines

Values:	ON	OFF

Default: OFF

Description: Split lines when longer than maximum allowed.

Details: When a text line is longer than 128K characters, the TEXTLINE input format truncates the line and either discards the remaining of the line (when this parameter is set to "OFF"), or processes the remainder of the line as a new line (when this parameter is set to "ON").

Example: -dQuotes:ON

iCheckpoint

Values:	checkpoint filename
Default:	not specified
Description:	Load and save checkpoint information to this file.
Details:	This parameter enables the "Incremental Parsing" feature that allows sequential executions of the same query to only process new log entries that have been logged since the last execution. For more information, see <u>Parsing Input Incrementally</u> .
Example:	-iCheckpoint:C:\Temp\myCheckpoint.lpc

TEXTLINE Input Format Examples

HTML Links

Return the lines in an HTML document that contain links to other pages:

LogParser "SELECT Text FROM http://www.microsoft.adatum.com WHERE Text LIKE '%href%''' -i:TEXTLINE

TEXTWORD Input Format

The TEXTWORD input format returns words from generic text files.

The TEXTWORD input format makes it possible to parse text files in any format not supported natively by Log Parser, and retrieve each word (i.e. each string delimited by whitespace characters) as a single field.

From-Entity Syntax Fields Parameters Examples

See also:

TEXTLINE Input Format TSV Input Format

TEXTWORD Input Format From-Entity Syntax

<from-entity> ::= <filename> [, <filename> ...] | http://<url> | STDIN

The <from-entity> specified in queries using the TEXTWORD input format is either:

- A comma-separated list of paths to text files, eventually including wildcards;
- The URL of a text file;
- The "STDIN" keyword, which specifies that the input data is available from the input stream (commonly used when piping command executions).

Examples:

FROM *.txt, \\MyServer\FileShare*.tsv

FROM http://www.microsoft.adatum.com/example.tsv

type data.txt | LogParser "SELECT * FROM STDIN" -i:TEXTWORD

TEXTWORD Input Format Fields

The input records generated by the TEXTWORD input format contain the following fields:

Name	Туре	Description
LogFilename	STRING	Full path of the file containing this word
Index	INTEGER	Word number
Text	STRING	Word

TEXTWORD Input Format Parameters

The TEXTWORD input format supports the following parameters:

iCodepage

Values:	codepage ID (number)
Default:	0
Description:	Codepage of the text file.
Details:	0 is the system codepage, -1 is UNICODE.
Example:	-iCodepage:1245

recurse

Values:	recursion level (number)
Default:	0
Description:	Max subdirectory recursion level.
Details:	0 disables subdirectory recursion; -1 enables unlimited recursion.
Example:	-recurse:-1

iCheckpoint

Values:	checkpoint filename
Default:	not specified
Description:	Load and save checkpoint information to this file.
Details:	This parameter enables the "Incremental Parsing" feature that allows sequential executions of the same query to only process new log entries that have been logged since the last execution. For more information,

see Parsing Input Incrementally.

Example: -iCheckpoint:C:\Temp\myCheckpoint.lpc

TEXTWORD Input Format Examples

Word Distribution

Return the distribution of words in the specified text file:

LogParser "SELECT Text, COUNT(*) FROM MyFile.txt GROUP BY Text O RDER BY COUNT(*) DESC" -i:TEXTWORD

TSV Input Format

The TSV input format parses tab-separated and space-separated values text files.

TSV text files, usually called "tabular" files, are generic text files containing values separated by either spaces or tabs.

This it also the format of the output of many command-line tools. For example, the output of the "netstat" tool is a series of lines, each line consisting of values separated by spaces:

Active Connections

Depending can Abarapplication the fixed line in a Jav file might be a "header" constaining the labor of the record field amond.corp.microsoft.co Theofollowing recommends a TSV file beginning with a header:

TCP GABRIEGI-M:microsoft-ds GABRIEGI-M.redmond.corp.microsoft.

2002 G295RIEGIAvpl1025 ion stareBRIEGI-M.redmond.corp.microsoft.com 20024ISTENINCWaiting for input

Among all the parameters supported by the HSV input format, the input format, the parameters supported by the HSV input format, the isolation of the parameters play a crucial role in providing the flexibility of the TSV input format on the format of the files being parsed.

TCP GABRIEGI-M:5000 GABRIEGI-M.redmond.corp.microsoft.com The is parate parameter specifies the character used as a separator between the fields in the files being paraget - M.redmond.corp.microsoft.com Some sext files, like the previous netstat example, use simple space characters as reparaton abaracters, and the files, like the resonable examples above (use tab characters.

The <u>msep</u> parameter specifies now many separator characters must appear for the characters to signify a field separator.

In the netstal example above, fields are separated by at least two space characters, while a single space character is allowed to appear in the value of a field (as is the case with the "Local Address" field name). On the other hand, in the previous tab-separated example file, fields are

separateGABREGABREAGHe/tal9@haracter.

BRIEGI-M:ntp *** p parameter specifies whether or not the fields in the input The tixed Sep parameter specifies whether or not the lields in the input UDP GABRIEGI-Minetbios-ng files are separated by a fixed number of separator characters. UDP GABRIEGI-Minetbios-dgm In the netstat example above, fields are separated by at least two space UDP GABRIEGI-Minetbios dgm

 $\frac{1-101}{2}$ $\frac{1-101}{2}$ chărăcters, but separator.

On the other hand, in the previous tab-separated example file, fields are separated by exactly a single tab character, and the presence of two consecutive tab characters signifies an empty field.

From-Entity Syntax Fields **Parameters Examples**

See also:

CSV Input Format TSV Output Format

TSV Input Format From-Entity Syntax

<from-entity> ::= <filename> [, <filename> ...] | http://<url> | STDIN

The <<u>from-entity</u>> specified in queries using the TSV input format is either:

- A comma-separated list of paths of TSV files, eventually including wildcards;
- The URL of a file in the TSV format;
- The "STDIN" keyword, which specifies that the input data is available from the input stream (commonly used when piping command executions).

Examples:

FROM LogFiles1*.txt, LogFiles2*.txt, \\MyServer\FileShare*.txt

FROM http://www.microsoft.adatum.com/MyTSVFiles/example.tsv

type data.tsv | LogParser "SELECT * FROM STDIN" -i:TSV

TSV Input Format Fields

The structure of the input records generated by the TSV input format is determined at run time, depending on the data being parsed, and on the values specified for the input format parameters.

The first two input record fields are fixed, and they are described in the following table:

Name	Туре	Description
Filename	STRING	Full path of the file containing this entry
RowNumber	INTEGER	Line in the file containing this entry

These two fields are then followed by the fields detected by the TSV input format in the file(s) being parsed. The number, names, and data types of the fields are determined by examining initially the input data according to the values specified for the input format parameters.

The number of fields detected by the TSV input format during the initial inspection phase dictates how the record fields will be extracted from the input data during the subsequent parsing stage.

If a line contains less fields than the number of fields established, the missing fields are returned as NULL values.

On the other hand, if a line contains more fields than the number of fields established, the extra fields are parsed as if they were part of the value of the last field expected by the TSV input format.

Number of Fields

The number of fields in an input record is determined by the input data and by the value of the <u>nFields</u> parameter.

When the "nFields" parameter is set to -1, the TSV input format determines the number of fields by inspecting the first line of the input

data, or the first line of the header file specified with the "iHeaderFile" parameter.

As an example, the following TSV file contains a variable number of fields:

Name City AreaCode

Jeff Redmond 425

When same doubt of the set to -1, this TSV file would yield three to file would yield three to be a contract of the set o

In this case, the extra fourth field in the second record would be parsed as part of the third "AreaCode" field, whose value would then be "206 98101".

When the "nFields" parameter is set to a value greater than zero, the TSV input format uses the specified value as the number of fields in the input data. Considering again the previous example file, parsing the file with the "nFields" parameter set to 4 would yield four fields.

Field Names

The names of the fields in an input record is determined by the input data and by the values of the <u>headerRow</u> and <u>iHeaderFile</u> parameters.

When the "headerRow" parameter is set to "ON", the TSV input format assumes that the first line in the file being parsed is a header containing the field names.

In this case, if the "iHeaderFile" parameter is left unspecified, the TSV input format extracts the field names from the header line.

On the other hand, if the "iHeaderFile" parameter is set to the path of a TSV file containing at least one line, then the TSV input format assumes that the specified file contains a header, parses its first line only, and extracts the field names from this line, ignoring the first line of the file being parsed.

If the number of field names extracted is less than the number of fields detected, the additional fields are automatically named "FieldN", with N being a progressive index indicating the field position in the input record.

Considering the previous example file, setting the "headerRow"

parameter to "ON" would cause the TSV input format to use the first line of the file as a header containing the field names.

With the "nFields" parameter set to -1, the TSV input format would detect three fields, whose names would be "Name", "City", and "AreaCode". On the other hand, with the "nFields" parameter set to 4, the TSV input format would detect four fields, named "Name", "City", "AreaCode", and "Field4".

When the "headerRow" parameter is set to "OFF", the TSV input format assumes that the file being parsed does not contain a header, and that its first line is the first data record in the file.

In this case, if the "iHeaderFile" parameter is set to the path of a TSV file containing at least one line, then the TSV input format assumes that the specified file contains a header, parses its first line only, and extracts the field names from this line.

On the other hand, if the "iHeaderFile" parameter is left unspecified, the fields are automatically named "Field*N*", with *N* being a progressive number indicating the field position in the input record.

As an example, the following TSV file does not contain a header line:

Jeff Redmond 425

Steve Seattle 206

When protection of the TSV file is the first data record in the file. In this case, the three fields would be named "Field1", "Field2", and "Field3".

Field Types

The <u>data type</u> of each field extracted from the input data is determined by examining the first n data lines, where n is the value specified for the <u>dtLines</u> parameter, in the following way:

- If all the non-empty field values in the first *n* lines are formatted as decimal numbers, then the field is assumed to be of the <u>REAL</u> type.
- If all the non-empty field values in the first *n* lines are formatted as integer numbers, then the field is assumed to be of the <u>INTEGER</u> type.
- If all the non-empty field values in the first *n* lines are formatted as

timestamps in the format specified by the <u>iTsFormat</u> parameter, then the field is assumed to be of the <u>TIMESTAMP</u> type.

• Otherwise, the field is assumed to be of the <u>STRING</u> type.

Empty field values are returned as <u>NULL</u> values.

TSV Input Format Parameters

The TSV input format supports the following parameters:

iSeparator

Values:	a single character spaces space tab
Default:	tab
Description:	Separator character between fields.
Details:	The "spaces" value instructs the TSV input format to consider any spacing character (space and tab) as a separator character.
Example:	-iSeparator:space

nSep

Values:	number of separators (number)
Default:	1
Description:	Number of separator characters between fields in the data records.
Details:	This parameter specifies how many separator characters must appear for the characters to signify a field separator. This parameter is usually set to a value greater than one when parsing space-separated text files in which field values can contain a single space character. In these cases, fields are usually separated by more than a single space character. When the "fixedSep" parameter is set to "OFF", the value of the "nSep" parameter is assumed to be the <i>minimum</i> number of separator characters signifying a field separator.

Example:	-nSep:2	
fixedSep		
Values:	ON OFF	
Default:	OFF	
Description:	Specifies whether or not the fields in the input TSV file(s) are separated by a fixed number of separator characters.	
Details:	When this parameter is set to "ON", the TSV input format assumes that the number of separator characters between the fields in the input data equals exactly the value specified for the "nSep" parameter. In this case, the presence of more separator characters signifies an empty value, which is returned as a NULL value. When this parameter is set to "OFF", the TSV input format assumes that the fields in the input data are separated by a variable number of separator characters, and the value of the "nSep" parameter is assumed to be the <i>minimum</i> number of separator characters signifying a field separator. In this case, additional separator characters are ignored and parsed as a single field separator, thus making it impossible for a value to be interpreted as a NULL value.	
Example:	-fixedSep:ON	

headerRow

Values: **ON | OFF**

Default: ON

Description: Specifies whether or not the input file(s) begin with a header line.

Details: When this parameter is set to "ON", the TSV input format assumes that each file being parsed begins with a header line, containing the labels of the fields in the file. If the "iHeaderFile" parameter is left unspecified, the TSV input format will use the field names in the first file's header as the names of the input record fields. If a value is specified for the "iHeaderFile" parameter, the TSV input format will ignore the header line in each file being parsed.

When this parameter is set to "OFF", the TSV input format assumes that the file(s) being parsed do not contain a header, and parses their first line as data records.

For more information on headers and field names, see <u>TSV Input Format Fields</u>.

Example: -headerRow:OFF

iHeaderFile

Values:	path to a TSV file
Values:	path to a TSV file

Default: *not specified*

Description: File containing field names.

Details: When parsing TSV files that do not contain a header line, the fields of the input records produced by the TSV input format are named "Field1", "Field2", ... To override this behavior and use meaningful field names, this parameter can be set to to the path of a TSV file containing a header line, causing the TSV input format to use the field names in the specified TSV file's header line as the names of the input record fields. Only the first line of the specified TSV file is parsed, and eventual additional lines are ignored. For more information on headers and field names, see <u>TSV Input Format Fields</u>.
Example: -iHeaderFile:"C:\My Folder\header.tsv"

nFields

Values:	number of fields (number)
Default:	-1
Description:	Number of fields in the data records.
Details:	This parameter specifies the number of fields in the input data. The special "-1" value specifies that the number of fields is to be deducted by inspecting the first line of input data. For more information on how the number of fields is determined, see <u>TSV Input Format Fields</u> .
Example:	-nFields:3

dtLines

Values:	number of lines (number)
Default:	100
Description:	Number of lines examined to determine field types at run time.
Details:	This parameter specifies the number of initial lines that the TSV input format examines to determine the data type of each input field. If the value is 0, all fields will be assumed to be of the STRING data type. For more information on how field data types are determined, see <u>TSV Input Format Fields</u> .
Example:	-dtLines:10

nSkipLines

Values: number of lines (number)

Default:	0
Description:	Number of initial lines to skip.
Details:	When this parameter is set to a value greater than zero, the TSV input format skips the first n lines of each input file before parsing its header line, where n is the value specified for this parameter.
Example:	-nSkipLines:5

lineFilter

Values:	+ - <any_string>[,<any_string>]</any_string></any_string>
Default:	not specified
Description:	Skip or consider only lines beginning with these strings.
Details:	When the value of this parameter begins with a "+" character, the TSV input format will only parse those lines beginning with one of the strings following the "+" character in the specified value. For example, the value "+Data:,Summary:" causes the TSV input format to parse only lines beginning with either "Data:" or "Summary:". When the value of this parameter begins with a "-" character, the TSV input format will ignore those lines beginning with one of the strings that follow the "-" character in the specified value. For example, the value "-Comment, Marker" causes the TSV input format to ignore lines beginning with either "Comment" or " Marker".
Example:	-lineFilter:"-Meta Data:, Summary:"

iCodepage

Values:	codepage ID (number)
Default:	0
Description:	Codepage of the TSV file.
Details:	0 is the system codepage, -1 is UNICODE.
Example:	-iCodepage:1245

iTsFormat

Values:	timestamp format
Default:	yyyy-MM-dd hh:mm:ss
Description:	Format of timestamp values in the input data.
Details:	This parameter specifies the date and/or time format used in the input data being parsed. Values of fields matching the specified format are returned as values of the <u>TIMESTAMP</u> data type. For more information on date and time formats, see <u>Timestamp Format</u> <u>Specifiers</u> .
Example:	-iTsFormat:"MMM dd, yyyy"

iCheckpoint

- Values: checkpoint filename
- Default: *not specified*
- Description: Load and save checkpoint information to this file.
- Details: This parameter enables the "Incremental Parsing" feature that allows sequential executions of the same query to only process new events that have been logged since the last execution. For more information, see <u>Parsing Input Incrementally</u>.

Example:

-iCheckpoint:C:\Temp\myCheckpoint.lpc

TSV Input Format Examples

NetStat output

Parse the output of a 'netstat' command:

netstat -a | LogParser "SELECT * FROM STDIN" -i:TSV -iSeparator:space -n Sep:2 -fixedSep:OFF -nSkipLines:3

URLSCAN Input Format

The URLSCAN input format parses log files created by the URLScan IIS filter.

URLScan is an ISAPI filter that allows administrators of web servers to restrict the kind of HTTP requests that the server will process. By blocking specific HTTP requests, the URLScan filter prevents potentially harmful requests from reaching the server and causing damage. The URLScan filter maintains a log file describing the actions taken when HTTP requests match the administrator-specified filters.

Log files created by the URLScan filter look like the following example:

[04-30-2002 - 17:09:48] ------ Initializing UrlScan.log ------[04-30-2002 - 17:09:48] -- Filter initialization time: [04-30-2002 - 17:09:48] -From-Entity-Syntax ------ UrlScan.dll Initializing ------124130-2002 - 17:09:49] UrlScan will return the following URL for rejected re puests: "/<Rejected-By-UrlScan>" 04-30-2002 - 17:09:49] URLs will be normalized before analysis. [04-30-2002 - 17:09:49] URL normalization will be verified. [04-30-2002 - 17:09:49] URLs must contain only ANSI characters. [04-30-2002 - 17:09:49] URLs must not contain any dot except for the file ext ension. [04-30-2002 - 17:09:49] URLs will be logged up to 128K bytes. [04-30-2002 - 17:09:49] Requests with Content-Length exceeding 30000000 will be rejected. [04-30-2002 - 17:09:49] Requests with URL length exceeding 260 will be reje cted. [04-30-2002 - 17:09:49] Requests with Query String length exceeding 4096 wi ll be rejected. [04-30-2002 - 17:09:49] Only the following verbs will be allowed (case sensiti ve): [04-30-2002 - 17:09:49] 'GET' [04-30-2002 - 17:09:49] Requests containing the following character sequence s will be rejected:

[04-30-2002 - 17:09:49] 'jj'

URLSCAN Input Format From-Entity Syntax

<from-entity> ::= URLSCAN |

<filename> [, <filename> ...]

The <from-entity> specified in queries using the URLSCAN input format is either the "URLSCAN" keyword or a comma-separated list of paths of URLScan log files.

When the "URLSCAN" keyword is used, the URLSCAN input format extracts the URLScan log configuration parameters from the UrlScan.ini configuration file and parses all the URLScan log files currently available in the URLScan log file directory.

Filenames can include wildcards (e.g. "URLSCAN\UrlScan*.log").

Examples:

FROM URLSCAN\UrlScan1.log, URLSCAN\UrlScan2.log

FROM \\MYMACHINE\URLSCAN\UrlScan*.log

FROM URLSCAN

URLSCAN Input Format Fields

The input records generated by the URLSCAN input format contain the following fields:

Name	Туре	Description
LogFilename	STRING	Full path of the log file containing this entry
LogRow	INTEGER	Line in the log file containing this entry
Date	TIMESTAMP	The date and time at which the request was served (local time)
ClientIP	STRING	The IP address of the client that made the request
Comment	STRING	The filter that matched the request and the action executed by URLScan
SiteInstance	INTEGER	The IIS virtual site instance number that served the request
Url	STRING	The HTTP request url

URLSCAN Input Format Parameters

The URLSCAN input format supports the following parameters:

iCheckpoint

Values:	checkpoint filename
Default:	not specified
Description:	Load and save checkpoint information to this file.
Details:	This parameter enables the "Incremental Parsing" feature that allows sequential executions of the same query to only process new log entries that have been logged since the last execution. For more information, see <u>Parsing Input Incrementally</u> .
Example:	-iCheckpoint:C:\Temp\myCheckpoint.lpc

URLSCAN Input Format Examples

Clients sending suspicious requests

Retrieve the DNS names of the clients that sent requests matching the URLScan filters:

LogParser "SELECT DISTINCT REVERSEDNS(ClientIP) FROM URLSCA N"

W3C Input Format

The W3C input format parses log files in the W3C Extended Log File Format.

Examples of log files in this format include:

- Personal Firewall log files
- Microsoft Internet Security and Acceleration Server (ISA Server) log files
- Windows Media Services log files
- Exchange Tracking log files
- Simple Mail Transfer Protocol (SMTP) log files

Log files in this format begin with some informative headers ("directives"), the most important of which is the "#Fields" directive, describing which fields are logged at which position in a log row.

After the directives, the log entries follow. Each log entry is a spaceseparated list of field values.

The following example shows a portion of a Personal Firewall W3C Extended Log File Format log file:

```
#Verson: 1.0
#Software: Microsoft Internet Connection Firewall
```

#Time Foldin die Folding und file und file with varying number and/or position
#Fieldsr flads time actions profer parsing pose for position actions profer parsing pose for position of the first size topflags to position. The first size topflags to position of the first size topflags tope of the first sis sis topflags to position. The first size topf

- - - -

See also:

IISW3C Input Format W3C Output Format

W3C Input Format From-Entity Syntax

<from-entity> ::= <filename> [, <filename> ...] | http://<url> | STDIN

The <from-entity> specified in queries using the W3C input format is either:

- A comma-separated list of paths of W3C Extended log files, eventually including wildcards;
- The URL of a file in the W3C Extended Log File Format;
- The "STDIN" keyword, which specifies that the input data is available from the input stream (commonly used when piping command executions).

Examples:

FROM LogFiles1\pf*.log, LogFiles2\pf*.log, \\MyServer\LoggingShare\pf*.lo g

FROM http://www.microsoft.adatum.com/MyLogFiles/example.log

type mylog.log | LogParser "SELECT * FROM STDIN" -i:W3C

W3C Input Format Fields

The structure of the input records generated by the W3C input format is determined at run time, depending on the input data.

The first two input record fields are fixed, and they are described in the following table:

Name	Туре	Description
LogFilename	STRING	Full path of the log file containing this entry
RowNumber	INTEGER	Line in the log file containing this entry

Following these two fields are all the fields declared by the first "#Fields" directive encountered in the input data.

The <u>data type</u> of each field extracted from the input data is determined by examining the first n log entries, where n is the value specified for the <u>dtLines</u> parameter, in the following way:

- If all the non-empty field values in the first *n* log entries are formatted as decimal numbers, then the field is assumed to be of the <u>REAL</u> type.
- If all the non-empty field values in the first *n* log entries are formatted as integer numbers, then the field is assumed to be of the <u>INTEGER</u> type.
- If all the non-empty field values in the first *n* log entries are formatted as timestamps in the "yyyy-MM-dd hh:mm:ss" format, then the field is assumed to be of the TIMESTAMP type. In particular, if a field value is formatted as a date in the "yyyy-MM-dd" format, then the value is returned as a <u>date-only TIMESTAMP</u> value. If the field value is formatted as a time of day in the "hh:mm:ss" format, then the value is returned as a <u>time-only TIMESTAMP</u> value.
- Otherwise, the field is assumed to be of the <u>STRING</u> type.

Empty values, represented by a hyphen (-) in the W3C Extended Log File Format, are returned as <u>NULL</u> values.

As an example, the following <u>help command</u> displays the input record structure determined by the W3C input format when parsing the specified Personal Firewall log file:

C:\>LogParser -h -i:W3C pfirewall.log

The structure displayed by this help command will be:

Fields:

LogFilename (RowNum	ber (I) dat	te (T) <u>All rights reserved</u> . dst-ip (S)
action (S)	protocol (S)	Src-ip (S)	dst-ip (S)
src-port (I)	dst-port (I)	size (I)	tcpflags (S)
tcpsyn (I)	tcpack (I)	tcpwin (I)	icmptype (S)
icmpcode (S)	info (S)		

W3C Input Format Parameters

The W3C input format supports the following parameters:

iCodepage

Values:	codepage ID (number)
Default:	0
Description:	Codepage of the log file.
Details:	0 is the system codepage, -1 is UNICODE.
Example:	-iCodepage:1245

dtLines

	Values:	number of lines (number)		
	Default:	10		
	Description:	Number of lines examined to determine field types at run time.		
	Details:	This parameter specifies the number of initial log lines that the W3C input format examines to determine the data type of the input record fields. If the value is zero, all fields will be assumed to be of the <u>STRING</u> data type. For more information on how field data types are determined, see <u>W3C Input Format Fields</u> .		
	Example:	-dtLines:50		
dQı	lotes			

Values:	ON OFF
Default:	OFF

Description:	Specifies that string values in the log are double- quoted.
Details:	Some W3C log files enclose string values within double-quote characters (").
Example:	-dQuotes:ON

separator

Values:	a single character	space	tab	auto
---------	--------------------	-------	-----	------

Default: auto

Description: Separator character between fields.

Details: Different W3C log files can use different separator characters between the fields; for example, Exchange Tracking log files use tab characters, while Personal Firewall log files use space characters. The "auto" value instructs the W3C input format to detect automatically the separator character used in the input log(s).

Example: -separator:tab

W3C Input Format Examples

Clients Sending Dropped Packets

Return all the clients that sent a packet dropped by Personal Firewall:

LogParser "SELECT DISTINCT src-ip FROM pfirewall.log WHERE action=' DROP'" -i:W3C

XML Input Format

The XML input format parses XML text files.

XML files (also called "XML documents") are hierarchies of *nodes*. Nodes can include other nodes, and each node can have a *node value* and a set of *attributes*.

For example, the following XML node has a value (in this instance, "Rome"), and a single attribute ("Population", whose value is, in this example, "3350000"):

```
<CITY Population='3350000'>Rome</CITY>
```

XML documents can be parsed in different ways, and the XML input format offers three distinct usages whose applicability depends on the structure of the documents, and on the structure of the information that needs to be extracted.

Note: The XML input format requires the Microsoft XML parser (MSXML) to be installed on the computer running Log Parser.

From-Entity Syntax Fields Parameters Examples

See also:

XML Output Format

XML Input Format From-Entity Syntax

<fromentity> ::= <document>[#<XPath>] [, <document>[#<XPath>] ...] <document> ::= <filename> | <url>

<document> ::= <filename> | <url>

The <from-entity> specified in queries using the XML input format is a comma-separated list of paths or URLs of XML files. Filenames or URLs can be optionally followed by an XPath that specifies which node(s) in the document are to be considered root node(s).

Filenames can include wildcards (e.g. "LogFiles\doc*.xml").

Examples:

FROM Document1.xml, http://blogs.msdn.com/MainFeed.aspx

FROM Document1.xml#/rss/channel/item, http://blogs.msdn.com/MainFeed.a spx#/rss/channel/item

XML Input Format Fields

The structure of the input records generated by the XML input format is determined at run time, depending on the document being parsed, and on the values specified for the input format parameters.

The XML input format parses an XML document by "visiting" the nodes in the document, and the input record fields are the **attributes** and **values** of the nodes that are visited by the XML input format.

By default, nodes are visited from the *document root*, that is, the single top-level node in an XML document that contains all the other nodes in the document.

However, by supplying an XPath in either the <u>from-entity</u> or as a value of the <u>rootXPath</u> parameter, users can specify that the document nodes are to be visited starting from the node(s) selected by the XPath.

Before parsing the XML document and return the input records, the XML input format initially examines the nodes found along the paths from the root node or from the node(s) selected by the user-supplied root XPath to the first *n* leaf nodes, where *n* is the value of the <u>dtNodes</u> parameter. During this phase, the XML input format creates a representation of the tree structure ("schema" tree) by merging nodes with the same name and hierarchical position. When completed, the schema tree contains one single instance of each node type, and each node contains an attribute set equal to the union of all the attributes found in the nodes of that type. At this moment, an input record field is created for each attribute belonging to a node type and for each node type having a value.

Once the schema tree has been determined and the input record structure has been created, the XML input format parses the XML document and generates input records, visiting the document nodes and extracting their values and attributes.

The XML input format implements three different algorithms to decide how document nodes will be visited. The three algorithms represent three different ways in which the information contained in an XML document can be retrieved, and the choice of an algorithm depends on the structure of the document and on the structure of the information that needs to be extracted.

Since different algorithms visit different sets of nodes, the choice of an algorithm affects which fields (i.e. which node attributes and values) will be contained in the input records.

Users can specify the algorithm to use through the <u>fMode</u> ("field mode") parameter, which can be set to "Branch", "Tree", or "Node".

Branch Field Mode

In this mode, input records contain the attributes and values of the nodes that are visited along all the possible paths from the document root or from the node(s) selected by the user-supplied root XPath to all the leaf nodes.

This mode is appropriate for documents in which each hierarchical level consists of nodes of the same type, as depicted in the following diagram:



In this structure, the root node contains only nodes of type "A", and each "A" node contains only nodes of type "B".

For example, the root of the following XML document contains "Continent" nodes only; each "Continent" node contains "Country" nodes only, and each "Country" node contains "City" nodes only:

<?xml version="1.0" ?>

<World>

This document can be thought of as containing six "entries", the leaf "City"on one and the information associated with each entry being contained in the nodes that are encountered along a path from the root nodectouthey leafung are='USA'>

In this grample the information about "Roma" includes the attributes and value of the ship and the "3350000" value of its/"Roma attribute), the attributes and value of its parent

"Country" node (the "Italia" value of the "CountryName" attribute), and the attributes tan Coaluey Naite glandpatent "Continent" node (the "Europe" value of the Vocontinent (lange" attribute).

<City> Toronto </City>

The scheme tree extracted from this example document specifies that the document root node contains nodes of the "Continent" type, and that nodes of the tree a "ContinentName" attribute. "Continent" nodes, in turn, contain nodes of the "Country" type, with a "CountryName" attribute intend to nodes of the "Country" type, with a "CountryName" attribute intend to nodes of the "Country" type, with a "CountryName" attribute intend to nodes of the "Country" type, with a "CountryName" attribute intend to nodes of the "Country" type, with a "CountryName" attribute intend to nodes of the "Country" type, and nodes of this type have a value, and a "Population" attribute. The input the contain four fields: p'Gontinent'Stated after the schema tree would thus contain four fields: p'Gontinent'Stated of "CountryName", "City", and "Population".

When using the Branch field mode, the XML input format generates an input record for each path from the document root node or from the node(s) selected by the user-supplied root XPath to all the leaf nodes. Each input record contains the attributes and values of the nodes encountered along the path: </World>





If a node does not specify an attribute that is contained in the attribute superset of the corresponding schema tree node, or if a node does not supply a value while the corresponding schema tree node specifies that at least one node of that type has a value, then the corresponding field value is set to NULL.

For example, parsing the above example XML document in "Branch" field mode would produce the following output:

ContinentName CountryName CityPopulationNorth America USARedmondNorth America USASan FranciscoTreet Fielde Mode anadaVancouver

In this Abdeicin battade ords contain the attributes and values of the nodes found in subtraces that and the all houses of distinct types.

Europe Italia Milano -

This mode is appropriate for documents in which a specific hierarchical level contains child nodes all having different types, as depicted in the following diagram:



In this structure, the root node contains only nodes of type "A"; each "A" node however contains nodes all having different types (a single "B"

node, a single "C" node, and a single "D" node).

For example, the root of the following XML document contains "Message" nodes; each "Message" node contains a single "From" node, a single "To" node, and a single "Body" node:

<?xml version="1.0" ?>

<Messages>

This document can be thought of as containing two "entries", the "Message" subtree and in the each entry being contained in eal/the modes in the subtree and in the nodes that are encountered along a path from the root node to the subtree root. In this example, 'the information about a message includes the attributes and/walkes of all the nodes included in the subtree ("From", "To", and "Body" nodes), and the attributes and values of all the nodes encountered along the path from the document root to the subtree root ("Date" attribute of the row of the subtree root).

<To> Gabriele </To>

The scheme tree extracted from this example document specifies that the document root node contains nodes of the "Message" type, and that nodes of this type have a "Date" attribute. "Message" nodes, in turn, contain nodes of the "From", "To", and "Body" types, each type having a node value.

The input records generated after the schema tree would thus contain four fields: "Date", "From", "To", and "Body".

When using the "Tree" field mode, the XML input format generates an input record for each subtree that includes all nodes of distinct types. Each input record contains the attributes and values of the nodes found in the subtrees, together with the attributes and values of the nodes encountered along the paths from the document root node or from the node(s) selected by the user-supplied root XPath to the subtree root nodes:



For example, parsing the above example XML document in "Tree" field mode would produce the following output:

Date	From	То	Body

2004-05-28 12:24:05 Gabriele Monica How's going?

While parsing an XML document in Tree, thanks, if a subtree is found containing multiple instances of the same node type, that subtree is "replicated" combinatorially to generate all the possible subtrees containing one single instance of each node type.

The following diagram depicts an XML document in which a subtree contains multiple instances of the same node type:



In this diagram, the "A" node contains one instance of the "B" node type, two instances of the "C" node type, and two instances of the "D" note type.

For example, the "Message" node in the following XML document contains a single "From" node, two "To" nodes, and two "Body" nodes:

```
<?xml version="1.0" ?>
```

<Messages>

This document can be thought of as a "compact" representation of four

diffevensage \$5age 2004-05-28T12:24:05'>

- From Gabrieleto/From in the "ENU" language;
- From "Gabriele" to "Jeff" in the "ITA" language;
- <To> Steve </To> From "Gabriele" to "Steve" in the "ENU" language; Body Language=ENU> Review ready? </Body> From "Gabriele" to "Steve" in the "ITA" language; From "Gabriele" to "Steve" in the "ITA" language;

</Message>
When using the "Tree" field mode, these "Message" subtrees are replicated combinatorially to generate all the possible subtrees containing one single instance of each of the "From", "To", and "Body" node types:



For example, parsing the above example XML document in "Tree" field mode would produce the following output:



XPath.

This mode is appropriate for situations in which the information to be retrieved is associated with a specific node type only. For example, the relevant information in the document depicted by the following diagram might be associated with "B" node types only:



When using the "Node" field mode, the XML input format generates an input record for each root node, either the document root or the node(s) selected by the user-supplied root XPath. Each input record contains the attributes and values of that node only:



For example, parsing the previous "Cities" example XML document in "Node" field mode specifying "/World/Continent/Country" as the root XPath would produce the following output:

CountryName
USA
Canada
Field Types
The data type of each field extracted from the schema tree is determined

in the following way:

- If all the non-empty field values (node values or attribute values) encountered while constructing the schema tree are formatted as decimal numbers, then the field is assumed to be of the <u>REAL</u> type.
- If all the non-empty field values (node values or attribute values) encountered while constructing the schema tree are formatted as integer numbers, then the field is assumed to be of the <u>INTEGER</u> type.
- If all the non-empty field values (node values or attribute values) encountered while constructing the schema tree are formatted as timestamps in the format specified by the <u>iTsFormat</u> parameter, then the field is assumed to be of the <u>TIMESTAMP</u> type.
- Otherwise, the field is assumed to be of the <u>STRING</u> type.

As an example, the following <u>help command</u> displays the input record structure determined by the XML input format when parsing the previous "Cities" example XML document:

```
C:\>LogParser -h -i:XML Cities.xml
```

The structure displayed by this help command will be:

Fields:

ContinentName (S) CountryName (S) City (S) Population (I) © 2004 Microsoft Corporation. All rights reserved.

XML Input Format Parameters

The XML input format supports the following parameters:

rootXPath

Values:	XPath query	
Default:	not specified	
Description:	XPath query of document node(s) to be considered root node(s).	
Details:	The node(s) selected by the specified XPath replace the document root node as the starting node(s) from which all the document nodes are visited.	
	Note : This parameter is ignored for XML documents whose filename or URL has been specified together with an optional XPath in the <u>from-entity</u> .	
	Note : The XPath specified for this parameter is case-sensitive. If an XPath is specified containing non-existing node or attribute names, or containing node or attribute names with the wrong capitalization, no root node is selected and an error is returned.	
Example:	-rootXPath:/World/Continent/Country	

fMode

- Values: Branch | Tree | Node | Auto
- Default: Auto
- Description: Algorithm to use when visiting the document nodes.

Details: For information on the "Branch", "Tree", and "Node" visit algorithms see <u>XML Input Format Fields</u>. The "Auto" value instructs the XML input format to determine automatically the best algorithm after inspecting the structure of the input document(s).

Example: -fMode:Tree

iTsFormat

Values:	timestamp format
Default:	yyyy-MM-dd?hh:mm:ss
Description:	Format of timestamp values in the document.
Details:	This parameter specifies the date and/or time format used in the document being parsed. Values of nodes or attributes matching the specified format are returned as values of the <u>TIMESTAMP</u> data type. For more information on date and time formats, see <u>Timestamp</u> <u>Format Specifiers</u> .
Example:	-iTsFormat:"MMM dd, yyyy"

dtNodes

Values:	number of leaf nodes (number)
Default:	-1

Description: Number of leaf nodes to be examined when determining the document structure.

Details: In order to determine the input document structure, the XML input format initially examines the nodes found along the paths from the root node or from the node(s) selected by the user-supplied root XPath to the first *n* leaf nodes, where *n* is the value specified for this parameter.
Specifying -1 causes the XML input format to examine *all* the nodes in the input document.

Example: -dtNodes:50

fNames

Values: **Compact | XPath**

Default: Compact

Description: Field naming schema.

Details: Specifying "Compact" causes the XML input format to create field names using the names of the corresponding nodes or attributes. If a field name is not unique, a sequential number is appended to the name to render it unique.

Example field names in the "Compact" mode are:

ContinentName

CountryName

Spacifying "XPath" causes the XML input format to created field names using the XPath queries for the corresponding nodes or attributes.

Example field names in the "XPath" mode are:

/World/Continent/@ContinentName /World/Continent/Country/@CountryName /World/Continent/Country/City -fNames:XPath /World/Continent/Country/City/@Population

Example:

XML Input Format Examples

MSDN BLogs Channel Titles

Display titles of current channels on MSDN BLogs:

LogParser "SELECT title FROM http://blogs.msdn.com/MainFeed.aspx#/rss/c hannel/item" -i:XML -fMode:Tree

Check Names from MBSA report

Display the checks in an MBSA report:

LogParser "SELECT Name FROM MYMACHINE.xml#/SecScan/Check" -fM ode:Node

Output Formats

Generic Text File Output Formats

- NAT: formats output records as readable tabulated columns.
- <u>CSV</u>: formats output records as comma-separated values text.
- <u>TSV</u>: formats output records as tab-separated or space-separated values text.
- <u>XML</u>: formats output records as XML documents.
- <u>W3C</u>: formats output records in the W3C Extended Log File Format.
- <u>TPL</u>: formats output records following user-defined templates.
- <u>IIS</u>: formats output records in the Microsoft IIS Log File Format.

Special-purpose Output Formats

- <u>SQL</u>: uploads output records to a table in a SQL database.
- <u>SYSLOG</u>: sends output records to a Syslog server.
- **DATAGRID**: displays output records in a graphical user interface.
- <u>CHART</u>: creates image files containing charts.

CHART Output Format

The CHART output format creates image files containing charts of the output record field values.

When using the CHART output format, output record fields must be of the <u>INTEGER</u> or <u>REAL</u> data types, in order for their values to be plotted in a chart.

The first field only can optionally be of the <u>STRING</u> or <u>TIMESTAMP</u> data types, in which case its values are used as the names of the categories on the X-axis of the chart.

The following example command creates a chart plotting the number of events logged in the System Event Log by each event source. The first field in the output records of this query is the name of the event source, and the CHART output format will use its values to label the categories along the X-axis of the chart. The second field in the output records is the number of events, which will be plotted on the chart:

LogParser "SELECT SourceName, COUNT(*) AS [Number of Events] INTO Events.gif FROM System GROUP BY SourceName ORDER BY [Number of The fast of the fast


Charts can also contain multiple series plotted from the values of different output record fields.

For example, the following command calculates the average, minimum, and maximum number of bytes served for each web page type:

LogParser "SELECT TO_UPPERCASE(EXTRACT_EXTENSION(cs-uri-ste m)) AS PageType, MIN(sc-bytes) AS Minimum, AVG(sc-bytes) AS Average, The resulting of the ballowing resulting of the second state of the second st



The CHART output format requires the Microsoft Office Web Components, which are generally installed with Microsoft Office 2000, Microsoft Office XP, and Microsoft Office 2003. In order to use the CHART output format, users must have a valid license of Microsoft Office for the computer executing the Log Parser query.

Configuration Scripts Into-Entity Syntax Parameters Examples

CHART Output Format Configuration Scripts

Charts created by the CHART output format can be customized by userprovided scripts in the JScript or VBScript languages that are executed by the CHART output format prior to generating the output image file.

These scripts can refer to two global objects which expose methods and properties that can be used to modify parameters such as the chart colors, the chart fonts, and many other attributes.

The two global objects available to configuration scripts are instances of the *chartSpace* and *chart* objects of the *Microsoft Office Web Components ChartSpace object model*, and they are named "chartSpace" and "chart", respectively.

For information on the *Office Web Components ChartSpace object model*, and on the *chartSpace* and *chart* objects, visit the <u>MSDN</u> <u>ChartSpace Object Model documentation</u>.

The following example script in the JScript language manipulates the *chartSpace* and *chart* objects to add a caption to the chart and to set the background color to the transparent color:

// Add a caption

chartSpace.HasChartSpaceTitle = true; Conafigurationserspta are used with the Geld Ret by the format by; specifying during the southe config parameter, as shown in the following action = chartSpace.Constants.chTitlePositionB ottom;

LogParser "SELECT SourceName, COUNT(*) AS [Number of Events] INTO EventssgeftFiR Davkgystend CoROUP BY SourceName ORDER BY [Number of Theorem Minge: hat wild work different to the second seco



CHART Output Format Into-Entity Syntax

<into-entity> ::= <filename>

The <u><into-entity></u> specified in queries using the CHART output format is the path to the output image file.

Examples:

INTO MyChart.gif

INTO \\COMPUTER01\Charts\Chart02.jpg

CHART Output Format Parameters

The CHART output format supports the following parameters:

chartType

Values:	name of chart type
Default:	Line
Description:	Chart type.
Details:	The set of available chart types depends on the version of the Microsoft Office Web Components installed on the lo computer. For a list of the available chart types, type the following <u>k</u> command from the command-line shell:
	LogParser -h -o:CHART

categories

Example:

Values: **ON | OFF | AUTO**

-chartType:Pie3D

Default: AUTO

Description: Display category labels along the category axis.

Details: When this parameter is set to "ON", the CHART output format uses the values of the first output record field to display category labels along the category axis. Setting this parameter to "AUTO" causes the CHART ou format to display category labels only when the first outp record field is of the <u>STRING</u> or <u>TIMESTAMP</u> data types Setting this parameter to "OFF" prevents the CHART out format from displaying category labels. Example: -categories:ON

maxCategoryLabels

8 5	
Values:	number
Default:	0
Description:	Maximum number of category labels displayed along the category axis.
Details:	This parameter is used to limit the number of category labels displayed along the category axis, in order to prev clutter in the output image. When this parameter is set to "0", the CHART output forr calculates the maximum number of category labels to display as a function of the dimensions of the target imag Setting this parameter to "-1" causes the number of category labels displayed along the category axis to be unlimited.
Example:	-maxCategoryLabels:20

legend

Default: AUTO

Description: Display a legend describing the series.

Details: When this parameter is set to "ON", the CHART output format displays a legend on the chart that describes the series being plotted.
Setting this parameter to "AUTO" causes the CHART our format to display a legend only when 2 or more series ar being plotted.
Setting this parameter to "OFF" prevents the CHART out format from displaying a legend.

Example:

-legend:ON

values

Values: **ON | OFF | AUTO**

Default: AUTO

Description: Display value labels.

Details: When this parameter is set to "ON", the CHART output format displays a label along each value being plotted, showing its numeric value.
Setting this parameter to "AUTO" causes the CHART our format to display value labels depending on the type of chart selected.
Setting this parameter to "OFF" prevents the CHART out format from displaying value labels.

Example: -values:ON

groupSize

Values:	widthxheight
Default:	640x480
Description:	Dimensions of the target image, in pixels.
Details:	This parameter specifies the width and height of the targ image, in pixels.
Example:	-groupSize:400x260

fileType

Default: AUTO

Description: Format of the output image file.

When this parameter is set to "AUTO", the CHART outpu
format determines the output image file format by inspec
the extension of the file specified for the into-entity.

Example: -fileType:JPG

config

Values:	comma-separated list of file paths
Default:	not specified
Description:	Configuration scripts to use for chart customization.
Details:	This parameter specifies a comma-separated list of scrip in the JScript or VBScript languages that can be used to further customize the chart generated by the CHART out format. For more information on configuration scripts, see <u>CHAF</u> <u>Output Format Configuration Scripts</u> .
Example:	- config:C:\MyScripts\MyConfig1.js,C:\MyScripts\MyConfig2

chartTitle

Default: Auto

Description: Title of the chart.

Details: When this parameter is set to "Auto" and the output reco contain 1 series only, the CHART output format uses the series' field name as the title of the chart.

Example: -chartTitle:"Bytes Per Page"

oTsFormat

Values: timestamp format

Default:	yyyy-MM-dd hh:mm:ss
----------	---------------------

Description: Format of timestamp values in the category labels.

Details: This parameter specifies the date and/or time format to u when formatting values of the <u>TIMESTAMP</u> data type to generate category labels. For more information on date and time formats, see <u>Timestamp Format Specifiers</u>.

Example: -oTsFormat:"MMM dd, yyyy"

view

- Values: **ON | OFF**
- Default: OFF

Description: Display chart image.

Details: Setting this parameter to "ON" causes the CHART output format to open a window displaying the generated output image file.

Example: -view:ON

CHART Output Format Examples

Top 20 URL's

Create a chart containing the TOP 20 URL's in the "www.margiestravel.com" web site:

LogParser "SELECT TOP 20 cs-uri-stem, COUNT(*) AS Hits INTO MyChart .gif FROM <www.margiestravel.com> GROUP BY cs-uri-stem ORDER BY Hits DESC" -chartType:Column3D -groupSize:1024x768

Bytes per Page Type

Create a pie chart with the distribution of bytes served for each page type:

LogParser "SELECT TO_UPPERCASE(EXTRACT_EXTENSION(cs-uri-ste m)) AS PageType, MUL(PROPSUM(sc-bytes),100.0) AS Bytes INTO Pie.gif FROM <1> GROUP BY PageType ORDER BY Bytes DESC" -chartType:Pie © 2004 Microsoft Corporation. All rights reserved. Exploded -chartTitle: Bytes per page type" -categories:off

CSV Output Format

The CSV output format writes output records as comma-separated values text.

The output of the CSV output format consists of multiple lines of text, one line for each output record.

Each line contains the values of the output record fields, separated by a comma (,) character. Depending on the value of the <u>oDQuotes</u> parameter, field values can be enclosed within double-quote characters (").

If enabled through the <u>headers</u> parameter, the first line in the output is a "header" that contains the names of the fields.

The following sample shows the output of the CSV output format when using the default values for its parameters:

EventID,SourceName,EventType,TimeGenerated

6009,EventLog,4,2004-04-18 18:48:04

Files; created with the QSM output format are suitable to be consumed by a lorg source on the phisation strate of the phisatio

```
7035,Service Control Manager,4,2004-04-18 18:48:27
7036,Service Control Manager,4,2004-04-18 18:48:27
```

```
174035, Staty i & Olantrol Manager, 4, 2004-04-18 18:48:27
```

```
P2035), Stelline Control Manager, 4, 2004-04-18 18:48:27
```

EX0330 Solution Control Manager, 4, 2004-04-18 18:48:27 7035, Service Control Manager, 4, 2004-04-18 18:48:27

```
7036,Service Control Manager,4,2004-04-18 18:48:27
```

See atso Service Control Manager,4,2004-04-18 18:48:27

```
7036,Service Control Manager,4,2004-04-18 18:48:27
```

```
Tags Steplite Compt Manager, 4, 2004-04-18 18:48:27
```

```
Chool (Service Control Manager, 4, 2004-04-18 18:48:27
```

```
-7036, Service Control Manager, 4, 2004-04-18 18:48:27
```

```
7035, Service Congrom Manager, & 2000, 24:18: 18:148:396s reserved.
```

```
7036,Service Control Manager,4,2004-04-18 18:51:26
```

```
7036, Service Control Manager, 4, 2004-04-18 18:51:29
```

6006,EventLog,4,2004-04-18 18:51:37

CSV Output Format Into-Entity Syntax

<into-entity> ::= <filename> | STDOUT

The <u><into-entity></u> specified in queries using the CSV output format is either:

- A filename;
- The "STDOUT" keyword, which specifies that the output data is to be written to the output stream (the console output).

The default into-entity for queries that do not specify an <u>INTO clause</u> is "STDOUT".

The CSV output format supports the <u>multiplex feature</u>, which can be enabled by specifying '*' wildcards in the into-entity filename. This feature allows output records to be written to different files depending on the values of their fields. For more information on the multiplex feature, see <u>Multiplexing Output Records</u>.

Examples:

INTO report.csv

INTO \\COMPUTER01\Reports\report.csv

INTO STDOUT

INTO Reports_*_*\Report*.csv

CSV Output Format Parameters

The CSV output format supports the following parameters:

headers

Values:	ON OFF AUTO
Default:	AUTO
Description:	Write a header line containing the field names.
Details:	 This parameter controls the CSV header line that is output at the beginning of each file. The possible values for this parameter are: ON: always write the header; OFF: never write the header; AUTO: write the header only when not appending to an existing file.
Example:	-headers:OFF
oDQuotes	

- Values: **ON | OFF | AUTO**
- Default: AUTO

Description: Enclose field values within double-quote characters (").

Details: This parameter controls whether or not the CSV output format should enclose field values within double-quote characters (").

The possible values for this parameter are:

- ON: always enclose field values within double-quote characters;
- OFF: never enclose field values within double-quote characters;

• AUTO: enclose within double-quote characters only those field values that contain comma (,) characters.

Example: -oDQuotes:ON

tabs

Values:	ON OFF
Default:	OFF
Description:	Write a tab character after each comma separator.
Details:	Setting this parameter to "ON" causes the CSV output format to write a tab character after each comma field separator, in order to improve readability of the CSV output. Note that using tabs between field values might generate output that is not compatible with certain spreadsheet applications.
Example:	-tabs:ON

oTsFormat

Values:	timestamp format
Default:	yyyy-MM-dd hh:mm:ss
Description:	Format of timestamp values in the output CSV data.
Details:	This parameter specifies the date and/or time format to use when formatting values of the <u>TIMESTAMP</u> data type. For more information on date and time formats, see <u>Timestamp Format Specifiers</u> .
Example:	-oTsFormat:"MMM dd, yyyy"

oCodepage

Values:	codepage ID (number)
Default:	0
Description:	Codepage of the output text.
Details:	0 is the system codepage, -1 is UNICODE.
Example:	-oCodepage:1245

fileMode

- Values: **0 | 1 | 2**
- Default: 1

Description: Action to perform when an output file already exists.

Details: This parameter controls the behavior of the CSV output format when the <u>into-entity</u> specifies directly or indirectly through the "multiplex" feature the name of a file that already exists. The possible values for this parameter are:

- Or existing files are enceded with the even
- 0: existing files are appended with the output;
- 1: existing files are overwritten with the output;
- 2: existing files are left intact, discarding the output.

Example:

-fileMode:0

CSV Output Format Examples

File Information

Create a CSV file containing information on the files contained in the specified directory:

LogParser "SELECT Path, Name, Size, Attributes INTO Files.csv FROM C:\T est*.*" -i:FS -o:CSV -recurse:0

Security Events

Retrieve the 10 latest events from the Security event log and write their information to a CSV file for each event ID:

LogParser "SELECT TOP 10 EventID, EventTypeName, Message INTO Even ts_*.csv FROM Security" -i:EVT -direction:BW -o:CSV

DATAGRID Output Format

The DATAGRID output format displays output records in a graphical user interface.

Output records are displayed in a scrollable grid that allows users to browse through the query results. Individual output records can be selected and copied to the clipboard as CSV-formatted data that can be pasted into another application.

The following screenshot shows the DATAGRID window displaying the results of a query:

EventID	TimeGenerated	EventCategoryName
4156	2004-02-02 09:49:45	Events
4156	2004-02-02 09:49:45	Events
100	2004-02-02 09:50:05	None
1500	2004-02-02 09:54:58	None
11406	2004-02-02 09:55:37	None
11708	2004-02-02 09:55:38	None
1017	2004-02-02 09:56:00	Setup
1000	2004-02-02 09:56:07	None

Controls in the DATAGRID user interface allow users to resize the window and the individual output record columns, and to change the properties of the font used to display the data.

Into-Entity Syntax Parameters Examples

See also:

NAT Output Format

DATAGRID Output Format Into-Entity Syntax

<into-entity> ::= DATAGRID

Queries using the DATAGRID output format are not required to specify an <u>INTO clause</u>. If an INTO clause is used, its <u><into-entity></u> must be specified as "DATAGRID".

Using the "DATAGRID" keyword in the <into-entity> allows Log Parser to select the DATAGRID output format automatically when no output format is explicitly specified.

Examples:

INTO DATAGRID

DATAGRID Output Format Parameters

The DATAGRID output format supports the following parameters:

rtp

Values:	number of rows
Default:	10
Description:	Rows to print before pausing.
Details:	The DATAGRID output format displays output records in batches made up of a number of rows equal to the value specified for this parameter. Once a batch of rows has been displayed, the "Next <i>n</i> rows" button is enabled, and the DATAGRID output format waits for the user to press the button before displaying the next batch of rows. Specifying "-1" for this parameter disables batching altogether.
Example:	-rtp:-1

autoScroll

- Values: **ON | OFF**
- Default: ON

Description: Automatically scroll window when new rows are output.

Details: When this parameter is set to "ON", the DATAGRID window scrolls down automatically whenever new output records are displayed, in order to position the display grid over the latest output records. Setting this parameter to "OFF" causes the grid position to remain unaltered when new output records are displayed.

This parameter is also accessible from the View menu

in the DATAGRID window.

Example: -autoScroll:OFF

DATAGRID Output Format Examples

Users' Job Titles

Retrieve users' job title breakdown from Active Directory:

LogParser "SELECT title, MUL(PROPCOUNT(*), 100.0) AS Percentage INT O DATAGRID FROM 'LDAP://MyUsername:MyPassword@mydomain/CN= Users,DC=mydomain,DC=com' WHERE title IS NOT NULL GROUP BY titl e ORDER BY Percentage DESC" -objClass:User **Registry Type Distribution**

Display the distribution of registry value types:

LogParser "SELECT ValueType, COUNT(*) FROM \HKLM GROUP BY Val ueType" -o:DATAGRID

IIS Output Format

The IIS output format writes output records in the Microsoft IIS Log File Format.

The following example shows a sample output file generated by the IIS output format:

192.168.1.1, -, 11/18/2003, 0:28:33, -, -, 192.168.1.100, 15, 194, 345, 304, -, G ET, /Default.htm, -,

Intop Eighty. Syntax 18/2003, 0:28:33, -, -, 192.168.1.100, 0, 139, 323, 304, -, G Perransienterss, -,

EXEMPLES.1, -, 11/18/2003, 0:28:33, -, -, 192.168.1.100, 0, 139, 334, 304, -, G ET, /images/address.gif, -,

192.168.1.1, -, 11/18/2003, 0:28:33, -, -, 192.168.1.100, 31, 2285, 273, 200, -, SEE also: bin/counts.exe, test=npa&style;=14,

192.168.1.2, -, 11/18/2003, 0:28:42, -, -, 192.168.1.100, 1828, 666, 442, 200, -,

192.168.1.2, -, 11/18/2003, 0:28:42, -, -, 192.168.1.100, 47, 2018, 463, 200, -, GET, /home@u2004thMicrosoft Corporation. All rights reserved.

192.168.1.2, -, 11/18/2003, 0:28:42, -, -, 192.168.1.100, 62, 8903, 308, 200, -, GET, /home/rules.htm, -,

IIS Output Format Into-Entity Syntax

<into-entity> ::= <filename> | STDOUT

The <u><into-entity></u> specified in queries using the IIS output format is either:

- A filename;
- The "STDOUT" keyword, which specifies that the output data is to be written to the output stream (the console output).

The default into-entity for queries that do not specify an <u>INTO clause</u> is "STDOUT".

The IIS output format supports the <u>multiplex feature</u>, which can be enabled by specifying '*' wildcards in the into-entity filename. This feature allows output records to be written to different files depending on the values of their fields. For more information on the multiplex feature, see <u>Multiplexing Output Records</u>.

Examples:

INTO inetsv1.log

INTO \\COMPUTER01\Logs\in040528.log

INTO STDOUT

INTO Logs_*_*\in*.log

IIS Output Format Parameters

The IIS output format supports the following parameters:

rtp

Values:	number of rows
Default:	10
Description:	Rows to print before pausing.
Details:	When writing to STDOUT, the IIS output format displays output records in batches made up of a number of rows equal to the value specified for this parameter. Once a batch of rows has been displayed, the IIS output format prompts the user to press a key to display the next batch of rows. Specifying "-1" for this parameter disables batching altogether.
Example:	-rtp:-1

oCodepage

Values:	codepage ID (number)
Default:	0
Description:	Codepage of the output text.
Details:	0 is the system codepage, -1 is UNICODE.
Example:	-oCodepage:1245

fileMode

Values:	0 1 2
Default:	1

Description: Action to perform when an output file already exists.

Details: This parameter controls the behavior of the IIS output format when the <u>into-entity</u> specifies directly or indirectly through the "multiplex" feature the name of a file that already exists.

The possible values for this parameter are:

- 0: existing files are appended with the output;
- 1: existing files are overwritten with the output;
- 2: existing files are left intact, discarding the output.

Example:

-fileMode:0

IIS Output Format Examples

W3C to IIS Conversion

Convert the specified W3C log file to an IIS log file:

LogParser "SELECT c-ip, cs-username, TO_DATE(TO_LOCALTIME(TO_TI MESTAMP(date, time))), TO_TIME(TO_LOCALTIME(TO_TIMESTAMP(d ate, time))), s-sitename, s-computername, s-ip, time-taken, sc-bytes, cs-bytes, s -c-status, sc-win32-status, cs-method, cs-uri-stem, cs-uri-query INTO inetsv1.lo g FROM extending "iiISW36-pills

NAT Output Format

The NAT output format writes output records in a readable tabulated column format.

The primary intended use of the NAT output format is to display output records to the console output.

This is the default output format selected by Log Parser when a command does not explicitly specify an output format and the query does not specify an <u>INTO clause</u>.

The following example shows a sample output generated by the NAT output format:

TimeGenerated	SourceName	EventID
2004-04-18 18:48:	04 EventLog	6009
12004-04-18-18:48: Into-Entity Syntax	04 EventLog	6005
-2004-04-18 18:48:	27 Service Control N	Manager 7024
2004-04-18 18:48:	27 Service Control N	Manager 7035
2004-04-18 18:48:	27 Service Control M	Aanager 7035
2004-04-18 18:48:	27 Service Control M	Aanager 7036
2004-04-18 18:48:	27 Service Control M	Aanager 7036
See also:18 18:48:	27 Service Control M	Aanager 7035
2004-04-18 48:48;	27-Service Control M	Aanager 7036
2004-04-18 18:48:	27 Service Control M	Manager 7035

NAT Output Format Into-Entity Syntax

<into-entity> ::= <filename> | STDOUT

The <u><into-entity></u> specified in queries using the NAT output format is either:

- A filename;
- The "STDOUT" keyword, which specifies that the output data is to be written to the output stream (the console output).

The default into-entity for queries that do not specify an <u>INTO clause</u> is "STDOUT".

The NAT output format supports the <u>multiplex feature</u>, which can be enabled by specifying '*' wildcards in the into-entity filename. This feature allows output records to be written to different files depending on the values of their fields. For more information on the multiplex feature, see <u>Multiplexing Output Records</u>.

Examples:

INTO report.txt

INTO \\COMPUTER01\Reports\report.txt

INTO STDOUT

INTO Reports_*_*\Report*.txt

NAT Output Format Parameters

The NAT output format supports the following parameters:

rtp

Values:	number of rows
Default:	10
Description:	Rows to print before pausing.
Details:	When writing to STDOUT, the NAT output format displays output records in batches made up of a number of rows equal to the value specified for this parameter. Once a batch of rows has been displayed, the NAT output format prompts the user to press a key to display the next batch of rows. Specifying "-1" for this parameter disables batching altogether.
Example:	-rtp:-1

headers

aucis	
Values:	ON OFF
Default:	ON
Description:	Print column headers.
Details:	This parameter enables or disables the column headers displayed before each batch of output rows.
Example:	-headers:OFF

spaceCol

Values:	ON OFF
Default:	ON

Description: Space columns uniformly.

Details: When this parameter is set to "ON", the NAT output format pads values with enough space characters to create columns having a uniform width within each batch of output rows. When this parameter is set to "OFF", the NAT output format displays unaligned values separated by a single space character.

Example:	-spaceCol:OFF	
----------	---------------	--

rAlign

Values:	ON OFF
Default:	OFF
Description:	Align columns to the right.
Details:	When this parameter is set to "ON", the NAT output format aligns values to the right side of each column. When this parameter is set to "OFF", values are aligned to the left side of each column.
Example:	-rAlign:ON

colSep

Values:	any string
Default:	single space character
Description:	Column separator.
Details:	This parameter specifies the separator to be used between the columns.
Example:	-colSep:", "

direct

Values: **ON | OFF**

Default: OFF

Description: Enable "direct mode".

Details: When "direct mode" is enabled, the NAT output format displays output records as they are made available, disabling the internal buffering mechanism used for column spacing and output row batching. In "direct mode" columns are not uniformly spaced, headers are printed only at the beginning of the output, and output records are displayed without interruption.

Example: -direct:ON	
---------------------	--

oCodepage

Values:	codepage ID (number)
Default:	0
Description:	Codepage of the output text.
Details:	0 is the system codepage, -1 is UNICODE.
Example:	-oCodepage:1245

fileMode

Values:	0 1 2
Default:	1
Description:	Action to perform when an output file already exists.
Details:	This parameter controls the behavior of the NAT output format when the <u>into-entity</u> specifies directly or indirectly through the "multiplex" feature the name of a file that already exists.

The possible values for this parameter are:

- 0: existing files are appended with the output;
- 1: existing files are overwritten with the output;
- 2: existing files are left intact, discarding the output.

Example:

-fileMode:0

NAT Output Format Examples

Ten Largest Files

Print the 10 largest files on the C: drive:

LogParser "SELECT TOP 10 * FROM C:*.* ORDER BY Size DESC" -i:FS

SQL Output Format

The SQL output format uploads output records to a table in a SQL database.

This output format can upload records to a table in any ODBC-compliant database, including Microsoft SQL Server and Microsoft Access databases.

When the target table does not already exist in the specified database, the SQL output format creates a table with as many columns as the number of fields in the <u>SELECT clause</u> of the query. In this case, the SQL type of each column is determined by the <u>data type</u> of the corresponding output record field, as described in <u>Column Type Mappings</u>.

If the target table already exists, the number of columns in the table must match exactly the number of fields in the SELECT clause of the query, and the SQL type of each column must be compatible with the data type of the output record field in the same position, as described in <u>Column</u> <u>Type Mappings</u>.

Column Type Mappings Into-Entity Syntax Parameters Examples
SQL Output Format Column Type Mappings

The following table shows the mappings between the <u>data types</u> of the query output record fields and the SQL types of the columns in the target table.

The column labeled "New Table" shows the SQL types declared for the table columns when the SQL output format creates the table. The column labeled "Existing Table" shows the SQL types that are compatible with the corresponding Log Parser data type when the SQL output format uploads records to an existing table.

Log Parser Data Type New Table Existing Table

INTEGER REAL	int real	int, bigint, smallint, tinyint, bit ¹ real, decimal, float
STRING) varchar(<i>n</i>), nvarchar(<i>n</i>), char
<u>TIMESTAMP</u>	datetime	datetime, smalldatetime, date, time
NULL	varchar	any type

Notes:

(1): when uploading to a field of the *bit* type, the target value is set to *true* when the INTEGER value is different than zero, and to *false* when the value is NULL or zero.

(2): the maximum length of new fields of the *varchar* type can be controlled through the <u>maxStrFieldLen</u> parameter.

© 2004 Microsoft Corporation. All rights reserved.

SQL Output Format Into-Entity Syntax

<into-entity> ::= <table_name>

The <u><into-entity</u>> specified in queries using the SQL output format is the name of the table where the results are to be uploaded to.

If the specified table does not already exist, the SQL output format creates a table with as many columns as the number of fields in the <u>SELECT clause</u> of the query. In this case, the SQL type of each column is determined by the <u>data type</u> of the corresponding output record field, as described in <u>Column Type Mappings</u>.

If the specified table already exists, the number of columns in the table must match exactly the number of fields in the SELECT clause of the query, and the SQL type of each column must be compatible with the data type of the output record field in the same position, as described in <u>Column Type Mappings</u>.

Examples:

INTO ReportTable

© 2004 Microsoft Corporation. All rights reserved.

SQL Output Format Parameters

The SQL output format supports the following parameters:

server

Values:	server name
Default:	
Description:	Name of the database server.
Details:	Setting a value for the "oConnString" parameter causes this parameter to be ignored.
Example:	-server:SQLREPORTS

database

Values:	database name
Default:	not specified
Description:	Name of the target database.
Details:	Setting a value for the "oConnString" parameter causes this parameter to be ignored.
Example:	-database:LogParserLogs

driver

Values:	ODBC driver name	
Default:	SQL Server	
Description:	Name of the ODBC driver to use.	
Details:	Setting a value for the "oConnString" parameter causes this parameter to be ignored.	
Example:	-driver:"Microsoft Access Driver (*.mdb)"	

dsn

Values:	DSN name
Default:	not specified
Description:	Name of the DSN to use.
Details:	This parameter can be used to specify a Data Source Name that contains information about the connection to the target database. Setting a value for the "oConnString" parameter causes this parameter to be ignored.
Example:	-dsn:"My DSN"

username

Values:	SQL username
Default:	not specified
Description:	Database username.
Details:	When this parameter is not specified, the SQL output format uses the current user's credentials through Windows Integrated Authentication. Setting a value for the "oConnString" parameter causes this parameter to be ignored.
	Note : For security reasons, values specified for this parameter are not persisted when using the Log Parser command-line <u>Defaults Override Mode</u> .
Example:	-username:MyDBUser
password	
Values:	SQL password
Default:	not specified

Description:	Database u	ser password.
--------------	------------	---------------

Details: Setting a value for the "oConnString" parameter causes this parameter to be ignored.

Note: For security reasons, values specified for this parameter are not persisted when using the Log Parser command-line <u>Defaults Override Mode</u>.

Example: -password:MyPassword

oConnString

- Values: connection string
- Default: *not specified*
- Description: ODBC connection string containing the parameters for th connection to the database.
- Details: Setting a value for this parameter causes the SQL output format to ignore any value set for the "server", "database "driver", "dsn", "username", and "password" parameters. The SQL output format does not enforce any syntax on the connection string. The value specified for this parameter is handed directly to the ODBC subsystem when initiating the connection to the database.

Note: For security reasons, values specified for this parameter that contain a username and/or a password are not persisted when using the Log Parser command-line <u>Defaults Override Mode</u>.

Example: -oConnString:"Driver={SQL Server};Server=MyServer;db=pubs;uid=sa;pwd=MyPassword

createTable

- Values: **ON | OFF**
- Default: OFF

- Description: Create a new table when the table specified in the <u>into-</u> <u>entity</u> does not exist.
- Details: When this parameter is set to "ON" and the target table does not already exist in the specified database, the SQI output format creates a table with as many columns as the number of fields in the <u>SELECT clause</u> of the query. this case, the SQL type of each column is determined by the data type of the corresponding output record field, as described in <u>Column Type Mappings</u>. When this parameter is set to "OFF" and the target table does not already exist in the specified database, the SQI output format generates an <u>error</u>, causing the currently executing query to abort.

Example: -createTable:ON

clearTable

Values:	ON OFF
Default:	OFF
Description:	Clear existing table before inserting new rows.
Details:	Setting this parameter to "ON" causes the SQL output format to delete existing rows in the target table before inserting the query output records.
Example:	-clearTable:ON

fixColNames

Values:	ON OFF
Default:	ON
Description:	Automatically remove invalid characters from column names when creating the target table.
Details:	When the "createTable" parameter is set to "ON" and the

target table does not already exist in the specified database, the SQL output format creates the table namir its columns with the names of the query output record fields. When this parameter is set to "ON", the SQL output format processes the field names and removes or substitutes those characters that are considered illegal b most databases, including space characters, parenthesy characters, and dash (-) characters.

Example: -fixColNames:OFF

maxStrFieldLen

- Values: number of characters
- Default: 255
- Description: Maximum number of characters declared for string columns when creating a table.
- Details: When the "createTable" parameter is set to "ON" and the target table does not already exist in the specified database, the SQL output format creates the table determining the SQL type of each column from the data type of the corresponding output record field, as described in <u>Column Type Mappings</u>. Columns corresponding to output record fields of the <u>STRING</u> data type are declared as SQL strings having a maximum length equal to the value specified for this parameter.

Example: -maxStrFieldLen:511

transactionRowCount

Values: number of rows

Default: 0

Description: Number of rows enclosed in a SQL transaction.

Details: When this parameter is set to "0", the SQL output format works in "auto commit" mode, where each single output record uploaded to the target table is automatically committed.
When this parameter is set to "-1", the SQL output forma initiates a SQL transaction when uploading the first outpur record, and commits or rollbacks the transaction after uploading the last record or when an error causes the query execution to abort.
Setting this parameter to any other value causes the SQ output format to create multiple SQL transactions, each containing a number of records equal to the specified value.

Example: -transactionRowCount:200

ignoreMinWarns

- Values: **ON | OFF**
- Default: ON

Description: Ignore minor warnings.

Details: When this parameter is set to "ON", the SQL output format ignores minor warnings that might occur while uploading records to the target table, including data truncation warnings and invalid escape character errors. When this parameter is set to "OFF", all minor warnings are reported as <u>warnings</u> when the query execution is complete.

Example: -ignoreMinWarns:OFF

ignoreIdCols

Values:	ON OFF
Default:	OFF

Description: Ignore "identity" columns in the target table.

When this parameter is set to "OFF" and the target table Details: specified in the into-entity already exists, the SQL output format expects a 1-to-1 match between the columns in th target table and the fields in the guery output records, regardless of whether or not any column in the target table is an "identity" column. In this case, the values of th output record fields will be uploaded to all the columns ir the table, including eventual "identity" columns. When this parameter is set to "ON" and the target table specified in the into-entity already exists, the SQL output format ignores "identity" columns in the target table, checking for a 1-to-1 match only between the non-identit columns and the fields in the query output records, and uploading output record field values to non-identity columns only.

Example:

-ignoreIdCols:ON

© 2004 Microsoft Corporation. All rights reserved.

SQL Output Format Examples

Upload Registry Values to a SQL table

Upload a portion of the registry into a newly-created SQL table:

LogParser "SELECT Path, KeyName, ValuleName INTO MyTable FROM \H KLM" -i:REG -o:SQL -server:MyServer -database:MyDatabase -driver:"SQL Server" -username:TestSQLUser -password:TestSQLPassword -createTable:O N

Upload IIS W3C log files to an Access database

Upload selected fields of an IIS W3C log file into an existing table in Microsoft Access:

LogParser "SELECT TO_TIMESTAMP(date, time), c-ip, cs-uri-stem, sc-statu s INTO MyTable FROM extend1.log" -i:IISW3C -o:SQL -oConnString:"Drive r={Microsoft Access Driver (*.mdb)};Dbg=C:\MyDB\MyDB.mdb;Uid=MyUs c_2004 Microsoft Corporation. All rights reserved. ername;Pwd=MyPassword

SYSLOG Output Format

The SYSLOG output format can be used to send messages to a Syslog server, to create text files containing Syslog messages, and to send Syslog messages to users.

The SYSLOG output format generates <u>messages</u> formatted according to the Syslog specifications described in RFC 3164.

Syslog messages consist of six parts, and the SYSLOG output format provides parameters that allow users to assign constants or output record fields to the different parts of a message.

The following example shows Syslog messages containing information gathered from the System event log:

<46>Apr 18 18:48:04 MYSERVER-M LogParser:EventLog: The Event log se rvice was started.

TheosxsLQQ 89489207 formatical theory polignally configured with Maxalog T sheven configuration file which describes at the rules used to forward

messages110 files. Sy \$1005\$#RV@R-MIUS@PSrser:EventLog: The Event log se rvice was stopped.

<134>Apr 18 19:20:23 MYSERVER-M LogParser:Ati HotKey Poller: The ser

Configuration Fibes 07 MYSERVER-M LogParser: EventLog: The Event log se

Performance in the service of the s

<46>Apr 18 19:33:17 MYSERVER-M LogParser:EventLog: The Event log se rvice was st@p@04 Microsoft Corporation. All rights reserved.

<134>Apr 19 07:01:57 MYSERVER-M LogParser:Ati HotKey Poller: The ser vice was started.

<46>Apr 19 07:01:41 MYSERVER-M LogParser:EventLog: The Event log se rvice was started.

<30>Apr 19 07:02:07 MYSERVER-M LogParser:Service Control Manager: T he Telephony service entered the running state.

SYSLOG Output Format Message Structure

The SYSLOG output format generates messages formatted according to the Syslog specifications described in RFC 3164. Syslog messages consist of six parts, and the SYSLOG output format provides parameters that allow users to assign constants or output record fields to the different parts of a message.

A sample Syslog message is formatted as follows:

<14>Nov 11 16:05:33 MYSERVER-M LogParser: The service was started.

This message consists of the following parts:

• **PRI**: <14>

The *PRI* part is bound with angle brackets and contains a decimal *Priority* value, which in turn is built as follows:

- The first 7 bits contain the <u>facility</u> value, describing the origin of the message;
- The last 3 bits contain the <u>severity</u> value, describing the importance of the message.
- HEADER: Nov 11 16:05:33 MYSERVER-M

The *HEADER* part consists of the following two elements:

- A <u>timestamp</u> value, indicating the local time at which the message was generated;
- A <u>hostname</u> value, indicating the host on which the message originated.
- **MSG**: LogParser: The service was started.

The *MSG* part consists of the following two elements:

- A tag value, indicating the name of the program or process that generated the message, followed by a colon character (":");
- A <u>content</u> value, containing the details of the message.

Facility

The facility value is represented by the upper 7 bits of the *priority* value in the *PRI* part of the message, and it describes the application or operating system component that originated the message. For a detailed list of the numeric values designated for well-known operating system components, refer to RFC 3164.

The following table shows the names assigned to the most common facility values:

Numerical Value	Facility Name
0	kern
1	user
2	mail
3	daemon
4	auth
5	mark
6	lpr
7	news
8	ииср
9	cron

10	auth2
11	ftp
12	ntp
13	logaudit
14	logalert
15	clock
16	local0
17	local1
18	local2
19	local3
20	local4
21	local5
22	local6
23	local7

In the previous example message, the priority value "14" indicates a facility value of 1 ("user").

The

<u>facility</u> parameter of the SYSLOG output format allows users to control the value of the facility field in the output messages.

This parameter can be set to any of the following values:

- A numeric value, such as "1" or "23";
- The name of a facility value, such as "user" or "local7";

 The name or the 1-based index of an output record field prepended with a dollar character ("\$"), such as "\$MyFacility" or "\$2". The specified output record field must be of either the INTEGER data type in which case its values are assumed to be numerical facility values, or of the STRING data type - in which case its values are assumed to be facility names among those described in the previous table. When an output record field value does not contain a recognized facility name or it contains a facility value greater than 23, the SYSLOG output format uses a default facility value of 1 ("user").

The following example query returns event messages from the System event log together with a "MyFacility" field that maps each event source to a Syslog facility name:

SELECT CASE SourceName

WHEN 'EventLog' THEN 'mark'

This query pars be it is to be retrieved from the "MyFacility py alugrated with the dauth"

WHEN 'NETLOGON' THEN 'logaudit'

LogPathsHEfNerAppQcaetionschoproSYSHENG 'looalf7Myconfig.conf -facility:\$MyFa cility ELSE 'local0'

The Systog messages generated by this command will look like the following agamples:

INTO SYSLOG

FRIMNSystem18:17:25 MYSERVER-M LogParser: The service was started.

<46>Nov 13 18:17:46 MYSERVER-M LogParser: The Event log service was s The the priority field of each of these messages contain the facility yalue provided synthe RMM Facility service ente red the running state.

<46>Nov 13 18:17:46 MYSERVER-M LogParser:The Event log service was s
topped.

Self Hov 13 18:17:46 MYSERVER-M LogParser: The service was started. The selverity value: Hore province alues in the granning of the message, and it describes the importance of the message. For a la: latter of the message and it describes the importance of the message. For a latter of the message and it describes the importance of the message. For a latter of the message and it describes the importance of the message. For a latter of the message and it describes the importance of the message. For a latter of the message and it describes the importance of the message. For a latter of the message and it describes the importance of the message. For a latter of the message and it describes the importance of the message. The Gold Now ing 112 to 12 45 how is the via the via the standard the

<134>Nov 13 18:17:46 MYSERVER-M LogParser: The service was started.

N460Perical Y4B1¢7:46SeverSieren Abreen LogParser: The Event log service was s tarted.

^Q30>Nov 13 18:17:46^eM¥^gERVER-M LogParser:The Telephony service ente

red the running state. 1	alert
2	crit
3	err
4	warning
5	notice
6	info
7	debug

For example, a priority value of "14" indicates a severity value of 6 ("info").

The

severity parameter of the SYSLOG output format allows users to control the value of the severity field in the output messages.

This parameter can be set to any of the following values:

- A numeric value, such as "1" or "7";
- The name of a severity value, such as "alert" or "debug";
- The name or the 1-based index of an output record field prepended with a dollar character ("\$"), such as "\$MySeverity" or "\$2". The specified output record field must be of either the INTEGER data type in which case its values are assumed to be numerical severity values, or of the <u>STRING</u> data type - in which case its values are assumed to be severity names among those described in the previous table. When an output record field value does not contain a recognized

severity name or it contains a severity value greater than 7, the SYSLOG output format uses a default severity value of 6 ("info").

The following example query returns event messages from the System event log together with a "MySeverity" field that maps each event type to a Syslog severity name:

SELECT CASE EventTypeName

WHEN 'Error event' THEN 'err'

This query party benaxe evend with the following command, which specifies that the specific for the specific or the specific or the "MySeventy" in the transmission of the specific of the spe

END AS MySeverity,

Log**Massagid**e:MyQuery.sql -o:SYSLOG -conf:Myconfig.conf -severity:\$MyS ENEFOySYSLOG

The System stages generated by this command will look like the following examples:

<14>Nov 13 21:42:15 MYSERVER-M LogParser:The Event log service was s tarted.

The lower Bibits 42f: the priority field of each afethese constances age a container the severity water provided by the "My Severity" of the container.

<14>Nov 13 21:42:15 MYSERVER-M LogParser:The Terminal Services servi ce was successfully sent a start control.

<12>Nov 13 21:42:15 MYSERVER-M LogParser:A request to suspend power **Timestamp** was defined by winlogon.exe.

The timestamp field in Witches HERIOUAL ogenerative the stand in witches say is some one of the stand it is usually formatted as follows:

Nov 11 16:05:33

If the first field in the query output records is of the

TIMESTAMP data type, the SYSLOG output format will use the field values to populate the timestamp field in the output messages. On the other hand, if the first field is not of the TIMESTAMP data type, the SYSLOG output format will use the current local time.

The following example query returns event messages from the System

event log together with the date and time at which the events have been generated:

SELECT TimeGenerated,

Message

The Systeg messages generated by this query will look like the following erronesystem

WHERE SourceName = 'EventLog'

<14>Apr 18 18:48:04 MYSERVER-M LogParser: The Event log service was st arted.

<14>Apr 18 18:51:37 MYSERVER-M LogParser: The Event log service was st opped.

Hostnappe 8 19:20:07 MYSERVER-M LogParser: Microsoft (R) Windows (R)

The hostname field indicates the server on which the message of the server on which the message of the server on the server of the server of the service was st arted.

The4>Apr 18 19:33:17 MYSERVER-M LogParser: The Event log service was st opped.

opped. hostName parameter of the SYSLOG output format allows users to <14>Apr 19 07:01:41 MY SERVER-M LogParser: Microsoft (R) Windows (R) centrol the value of the hostname field in the output messages. 5.01. 2600 Service Pack 1 Uniprocessor Free This parameter can be set to any of the following values: <14 Apr 19 07:01:41 MY SERVER-M LogParser. The Event log service was st

• a The "localhost" keyword, specifying that the field should be populated

< With ABE 1969: 29:11 MAY SERVER-M LogParser: The Event log service was st

- \bullet_0 "MYCOMPUTER":
- The name or the 1-based index of an output record field prepended with a dollar character ("\$"), such as "\$MyHostname" or "\$2". The specified output record field must be of the STRING data type, and its values will be used to populate the hostname field in the output messages.

When no value is specified for the "hostName" parameter, the hostname field is automatically populated with the local computer name.

The following example query returns event messages from the System event log of different computers, together with the computer name on which the event originated:

SELECT Message,

ComputerName

This gue is can be executed with the following command, which specifies that the host server field of each, own server sage is the be watris end field is the top output record field:

LogParser file:MyQuery.sql -o:SYSLOG -conf:Myconfig.conf -hostName:\$2

The Syslog messages generated by this command will look like the following examples:

<14>Nov 13 22:07:11 MYSERVER03 LogParser:Microsoft (R) Windows (R) 5.01. 2600 Service Pack 1 Uniprocessor Free.

<14>Nov 13 22:07:11 MYSERVER03 LogParser: The Event log service was st arted.

Tag4>Nov 13 22:07:11 MYSERVER01 LogParser:The Terminal Services servi

The way step shull ates the that he of the program or process that generated the message. SERVER02 LogParser: The Network Connections s ervice was successfully sent a start control.

The4>Nov 13 22:07:11 MYSERVER01 LogParser:The Terminal Services servi ce entered the running state.

processive parameter of the SYSLOG output format allows users to control the value of the tag field in the output messages.

This parameter can be set to any of the following values: 14 Nov 13 22:07:11 MYSERVER02 LogParser: The SSDP Discovery Servic • escretion of the set of the second service of the second second service of the second seco

<"IMYR609952:07:11 MYSERVER03 LogParser: The SSDP Discovery Servic

• e The inamesouthes if blass echindex of an indutput record field prepended with a dollar character ("\$"), such as "\$MyProgram" or "\$2". The specified output record field must be of the <u>STRING</u> data type, and its values will be used to populate the tag field in the output messages.

When no value is specified for the "processName" parameter, the tag field is automatically populated with "LogParser:".

Content

The content field contains the details of the message, and its value is built by the SYSLOG output format by concatenating the values of all the

output record fields, excluding those fields that are used for the values of the

facility, severity, timestamp, hostname, and tag message fields.

The following example query returns information from the System event log:

SELECT SourceName,

EventTypeName,

The System message grassing enerated by this query will look like the following examples age

INTO SYSLOG

FROMOSystem2:27:17 MYSERVER-M LogParser:EventLog Information even t None Microsoft (R) Windows (R) 5.01. 2600 Service Pack 1 Uniprocessor Fr

ee. © 2004 Microsoft Corporation, All rights reserved. <14>Nov 13 22:27:17 MYSERVER-M LogParser: EventLog Information even t None The Event log service was started.

<14>Nov 13 22:27:17 MYSERVER-M LogParser:Service Control Manager Er ror event None The Computer Browser service terminated with service-specifi c error 2550 (0x9F6).

<14>Nov 13 22:27:17 MYSERVER-M LogParser:EventLog Information even t None The Event log service was stopped.

<14>Nov 13 22:27:17 MYSERVER-M LogParser:Ati HotKey Poller Informat ion event None The service was started.

<14>Nov 13 22:27:17 MYSERVER-M LogParser:EventLog Information even t None Microsoft (R) Windows (R) 5.01. 2600 Service Pack 1 Uniprocessor Fr ee.

<14>Nov 13 22:27:17 MYSERVER-M LogParser:EventLog Information even t None The Event log service was started.

<14>Nov 13 22:27:17 MYSERVER-M LogParser:EventLog Information even t None The Event log service was stopped.

SYSLOG Output Format Configuration Files

Messages generated by the SYSLOG output format can be forwarded to any of the following three possible destinations:

- A Syslog server;
- A text file;
- A user, through the Windows alerter and messenger services.

The <u>conf</u> parameter of the SYSLOG output format allows users to specify a configuration file resembling the standard "syslog.conf" file that describes the rules used to forward messages to different destinations. These rules associate values of the <u>facility</u> and <u>severity</u> message fields with specific Syslog servers, text files, or users.

Each line in a configuration file is either a comment beginning with the pound character ("#"), or a *configuration entry*. Configuration entries have the following syntax:

<config_entry></config_entry>	::=	<selector> <action></action></selector>
<selector></selector>	::=	<facilities>.<severity></severity></facilities>
<facilities></facilities>	::=	<facility>[,<facility>]</facility></facility>
<facility></facility>	::=	kern user mail daemon auth mark lpr news uucp cron auth2 ftp ntp logaudit logalert clock local0 local1 local2 local3 local4 local5 local6 local7 *
<severity></severity>	::=	emerg alert crit err warning notice info debug
<action></action>	::=	<send_server> <send_file> </send_file></send_server>

<send_user>

<send_server></send_server>	::=	@ <server_name>[:<port>]</port></server_name>
<send_file></send_file>	::=	<filepath> STDOUT</filepath>
<send_user></send_user>	::=	<user_name></user_name>

A configuration entry is composed of a selector and an action, separated by spaces or tab characters.

A selector is a comma-separated list of facility names followed by a dot (".") and followed by a severity name. The special "*" wildcard means "all facilities".

Messages whose facility is included in the selector's set of facilities and whose severity is greater than or equal to the selector's severity are forwarded to the destination specified in the action.

An action can specify any of the following destinations:

- The name or address of a Syslog server, preceded by an at character ("@") and optionally followed by a port number; when no port number is specified, the SYSLOG output format will use port 514;
- The path of an output filename;
- The STDOUT keyword, which specifies that the output data is to be written to the output stream (the console output);
- The name of a user.

The following example shows a SYSLOG output format configuration file:

#

Sample SYSLOG output format configuration file T#is configuration file defines the following rules:

• a Messages from @MeY'& EIRV Faculity with a severity greater than or equal *toleberg" are forw & HO2O to The "MYSERVER01" Syslog server on port *51140, C:\MyLogs\Infos.txt

- kAthmessages having Lasser verity greater than or equal to "debug" are logist of last the geon and logist of the second second
- All messages having a severity greater than or equal to "info" are written to the "C:\MyLogs\Infos.txt" text file;
- Messages from the "kern" facility with a severity greater than or equal to "emerg" are sent to the "MYUSER" user;
- Messages from the "local0" or "local1" facilities with a severity greater than or equal to "emerg" are forwarded to the Syslog server with address 192.168.1.100 on port 515.

Messages matching more than one rule are forwarded to all the specified destinations. For example, with the above configuration file, messages having a severity greater than or equal to "debug" are both displayed in the console output and written to the "C:\MyLogs\Infos.txt" text file.

Actions can also be specified in the <u>into-entity</u> of the query. These actions are processed as rules having a selector that matches all messages, with a "*" facility value and an "emerg" severity value.

© 2004 Microsoft Corporation. All rights reserved.

SYSLOG Output Format Into-Entity Syntax

<into-entity></into-entity>	::=	<action> [, <action>] SYSLOG</action></action>
<action></action>	::=	<send_server> <send_file> <send_user></send_user></send_file></send_server>
<send_server></send_server>	::=	@ <server_name>[:<port>]</port></server_name>
<send_file></send_file>	::=	<filepath> STDOUT</filepath>
<send user=""></send>		

The <<u>into-entity></u> specified in queries using the SYSLOG output format is either the "SYSLOG" keyword, which specifies that messages should be forwarded according to the rules in the <u>configuration file</u> specified for the <u>conf</u> parameter, or a comma-separated list of actions, where each action is either:

- The name or address of a Syslog server, preceded by an at character ("@") and optionally followed by a port number; when no port number is specified, the SYSLOG output format will use port 514;
- The path of an output filename;
- The STDOUT keyword, which specifies that the output data is to be written to the output stream (the console output);
- The name of a user, to which Syslog messages will be sent through the Windows alerter and messenger services.

When a configuration file has been specified through the "conf" parameter, queries are allowed to not provide an <u>INTO clause</u> at all; if an INTO clause is used, its into-entity must be specified as "SYSLOG".

When a configuration file has not been specified, the INTO clause is mandatory and it must contain at least one valid action.

Actions specified in the into-entity are processed as <u>configuration rules</u> having a selector that matches all messages, with a "*" facility value and an "emerg" severity value.

Examples:

INTO SYSLOG

INTO @MYSERVER02:515

INTO \\COMPUTER01\Reports\report.txt

INTO MYUSER

INTO @MYSERVER01, C:\MyLogs\Infos.txt, STDOUT, MYUSER, @192.1 68.1.100:515

© 2004 Microsoft Corporation. All rights reserved.

SYSLOG Output Format Parameters

The SYSLOG output format supports the following parameters:

conf

Values:	file path
Default:	not specified
Description:	Syslog configuration file.
Details:	This parameter specifies the path to a <u>configuration file</u> that describes the rules used to forward messages to different destinations. When this parameter is used, queries are allowed to not provide an <u>INTO clause</u> at all; if an INTO clause is used, its into-entity must be specified as "SYSLOG". For more information on configuration files, see <u>SYSLOG Output Format Configuration Files</u> .
- 1	

Example: -conf:C:\mysyslog.conf

severity

Values:	<numeric_value> <name> \$<field_name> \$<field_index></field_index></field_name></name></numeric_value>
Default:	info
Description:	Message severity level.
Details:	 This parameter controls the value of the severity field of the output messages. The possible values for this parameter are: A numeric value, such as "1" or "7"; The name of a severity value, such as "alert" or "debug";

• The name or the 1-based index of an output record

field prepended with a dollar character ("\$"), such as "\$MySeverity" or "\$2". The specified output record field must be of either the INTEGER data type - in which case its values are assumed to be numerical severity values, or of the <u>STRING</u> data type - in which case its values are assumed to be severity names among those described in the previous table. When an output record field value does not contain a recognized severity name or it contains a severity value greater than 7, the SYSLOG output format uses a default severity value of 6 ("info").

For more information on the severity field of the output messages, see <u>SYSLOG Output Format Message</u> <u>Structure</u>.

Examples:	-severity:1
	-severity:alert
	-severity:\$MySeverity
	-severity:\$2

facility

Values:	<numeric_value> <name> \$<field_name> \$<field_index></field_index></field_name></name></numeric_value>
Default:	user
Description:	Message facility.
Details:	 This parameter controls the value of the <u>facility</u> field of the output messages. The possible values for this parameter are: A numeric value, such as "1" or "23"; The name of a facility value, such as "user" or "local7"; The name or the 1-based index of an output record

field prepended with a dollar character ("\$"), such as

"\$MyFacility" or "\$2". The specified output record field must be of either the INTEGER data type - in which case its values are assumed to be numerical facility values, or of the STRING data type - in which case its values are assumed to be facility names among those described in the previous table. When an output record field value does not contain a recognized facility name or it contains a facility value greater than 23, the SYSLOG output format uses a default facility value of 1 ("user").

For more information on the facility field of the output messages, see <u>SYSLOG Output Format Message</u> <u>Structure</u>.

Examples:	-facility:23
	-facility:local7
	-facility:\$MyFacility
	-facility:\$2

oTsFormat

Values:	timestamp format
Default:	MMM dp hh:mm:ss
Description:	Format of the timestamp field.
Details:	This parameter specifies the format of the <u>timestamp</u> field of the output messages. For more information on date and time formats, see <u>Timestamp Format Specifiers</u> . For more information on the timestamp field of the output messages, see <u>SYSLOG Output Format</u> <u>Message Structure</u> .
E	

Example: -oTsFormat:"MMM dd, yyyy"

hostName

Values: **localhost** | <name> | \$<field_name> | \$<field_index>

Default: localhost

Description: Value of the hostname field.

Details: This parameter controls the value of the <u>hostname</u> field of the output messages.

The possible values for this parameter are:

- The "localhost" keyword, specifying that the field should be populated with the local computer name;
- A generic string indicating the desired host name, such as "MYCOMPUTER";
- The name or the 1-based index of an output record field prepended with a dollar character ("\$"), such as "\$MyHostname" or "\$2". The specified output record field must be of the <u>STRING</u> data type, and its values will be used to populate the hostname field in the output messages.

For more information on the hostname field of the output messages, see <u>SYSLOG Output Format</u> <u>Message Structure</u>.

-	-hostName:MYCOMPUTER
	-hostName:\$MyHostname
	-hostName:\$2

processName

Values: <name> | \$<field_name> | \$<field_index>

Default: LogParser:

Description: Value of the tag field.

Details: This parameter controls the value of the tag field of the

output messages.

The possible values for this parameter are:

- A generic string indicating the desired tag field value, such as "MyReports";
- The name or the 1-based index of an output record field prepended with a dollar character ("\$"), such as "\$MyProgram" or "\$2". The specified output record field must be of the <u>STRING</u> data type, and its values will be used to populate the tag field in the output messages.

For more information on the tag field of the output messages, see <u>SYSLOG Output Format Message</u> <u>Structure</u>.

	-processName:MyReports
	-processName:\$MyProgram
	-processName:\$2

separator

- Values: any string | **space** | **tab**
- Default: space
- Description: Separator between fields.
- Details: This parameter controls the separator to be used between the message fields. The "tab" keyword causes the SYSLOG output format to use a single tab character between the fields, while the "space" keyword causes the SYSLOG output format to use a single space character.

Example: -separator:tab

maxPacketSize

Values: number of bytes

Default: 1024

Description: Maximum message size.

This parameter controls the maximum size of the
messages generated by the SYSLOG output format.
Messages whose size exceeds the value specified for
this parameter are either truncated or discarded,
depending on the value of the "discardOversized"
parameter.

discardOversized

Values:	ON OFF
Default:	OFF
Description:	Discard oversized messages.
Details:	When this parameter is set to "ON", the SYSLOG output format discards messages whose size exceeds the value specified for the "maxPacketSize" parameter. When this parameter is set to "OFF", the SYSLOG output format truncates oversized messages to the size specified with the "maxPacketSize" parameter.
Example:	-discardOversized:ON

protocol

Values:	UDP TCP
Default:	UDP
Description:	Protocol used for transmission.
Details:	This parameter specifies the protocol to use when sending messages to Syslog servers.

Example: -prote

-protocol:TCP

sourcePort

Values:	port number *
Default:	*
Description:	Source port to use for transmission.
Details:	This parameter specifies the source port to use when sending messages to Syslog servers. Specifying "*" causes the SYSLOG output format to choose any available port number.
Example:	-sourcePort:514

ignoreDspchErrs

Values:	ON OFF
Default:	OFF
Description:	Ignore dispatch errors.
Details:	Setting this parameter to "ON" causes the SYSLOG output format to buffer errors occurring while transmitting messages to Syslog servers or users, reporting all the errors as <u>warnings</u> when the query execution has completed. Setting this parameter to "OFF" causes the SYSLOG output format to report errors as they occur, aborting the execution of the query.
Example:	-ignoreDspchErrs:ON
- J	

oCodepage

Values: codepage ID (number)

Default: 0

Description: Codepage of the output message text.

Details: 0 is the system codepage, -1 is UNICODE.

Example: -oCodepage:1245

© 2004 Microsoft Corporation. All rights reserved.

SYSLOG Output Format Examples

Export System Event Log

Export events from the System event log to a Syslog server and to a local file:

SELECT TimeGenerated,

CASE SourceName

This qweffeqane benever utenevith the following command:

WHEN 'Service Control Manager' THEN 'daemon'

LogPathselfINelPyIncQuefEStallpo:SYSLOG -facility:\$MyFacility -severity:\$MyS everitW-H&SNNKreed&CoohTHHENNanth'

The output will be the following sample:

WHEN 'Application Popup' THEN 'local7'

<46>AbSE8'lb&a40!04 MYSERVER-M LogParser:EventLog: The Event log se rvice was started. Facility,

<30 CADE A Store By Brand By De Marge RVER-M Log Parser: Service Control Manager: T he Telephony 'sanoice venter at the remaining state.

IIS469 MEMERAN WOLF AND AN AND A TO BE AND

Seife endermation Isered's Tota Systers server: <134 EAGE18nf0:20:23 MYSERVER-M LogParser: Ati HotKey Poller: The ser

vice ENDstasted vSeverity SELECT TO THMESTAMP(date, time), 2425 Sound ReTWAM0.7 MYSERVER-M LogParser:EventLog: The Event log se THIS ON CALENDER FOR THE FEAT

Augustic with the following command: MYSERVER-M LogParser:Service Control Manager: T <30 **Magss 12**e19;20

WERed dred the running state Werverysql o:SYSLOG -facility:logalert -severity:\$MySeveri 33:17 MYSERVER-MLogParser:EventLog: The Event log se aviteoscialing:Hpsonassevame:IIS:

The messages will look like the following samples: <134>Apr 19 07:01:57 MYSERVER-M LogParser:Ati HotKey Poller: The ser

XYT5) MANYSER: V8:RB4MYSE ER04 IIS:/images/tibg.gif 404 R-M LogParser:EventLog: The Event log se FRISH 00:28:4

04 IIS:/images/tibg.gif 404 A LogParser:Service Control Manager: T 94 IIS:/aa.css 404 303Apr 19860 he Telephony se

<115>Nov 18 00:29:02 M SERVER04 IIS:/images/tibg.gif 404 <115>Nov 18 00:29:04 MYSERVER04 IIS:/gorice/rulesinfo.nsf 403 <115>Nov 18 00:29:05 MYSERVER04 IIS:/_vti_inf.html 404 <112>Nov 18 00:29:05 MYSERVER04 IIS:/_vti_bin/shtml.dll 500 <115>Nov 18 00:31:51 MYSERVER04 IIS:/na/index.html 404

TPL Output Format

The TPL output format writes output records formatted according to userdefined <u>templates</u>.

Templates are text files divided into three sections - a *header*, a *body*, and a *footer* - containing variables that refer to the values and names of the output record fields.

During the output generation stage, the TPL output format substitutes the variables with the values of the output record fields, generating text files formatted according to the user specifications.

The flexibility of the TPL output format allows users to generate HTML files, XML files, and generic text files in almost any format.

Template Files Into-Entity Syntax Parameters Examples

© 2004 Microsoft Corporation. All rights reserved.
TPL Output Format Template Files

Template files are divided into three sections: an optional *header* section that is written once at the beginning of the output, a *body* section that is written repeatedly for each output record, and an optional *footer* section that is written once at the end of the output.

The body section can contain special <u>variables</u> that are substituted at run time with values computed during the execution of the query, such as values and names of output record fields, and the number of fields in the output records.

The header and footer sections can contain the same variables available to the body section, except for those that refer to values of output record fields.

Template files can be specified in two different ways: as *raw format* templates, or as *structured format* templates.

Raw Format Templates

In the raw format, the three template sections are specified as three different files.

The template file containing the body section is specified using the tpl parameter, while the optional header and footer sections are specified with the tplHeader and tplFooter parameters, respectively.

The following is a sample raw format template file containing the body section:

The Url %cs-uri-stem%, requested by %c-ip%, took %time-taken% millisecon ds to execute.

The formatted according to the template file:

LogParser "SELECT * INTO out.txt FROM extend1.log" -o:TPL -tpl:mytempl ate.tpl

The resulting output will look like the following example:

The Url /default.htm, requested by 192.168.1.102, took 24 milliseconds to exe cute.

StructuredeFiedmaal4Tebmblatebock.

The Url /mydocuments/index.html, requested by 192,168,1,104, took 134 milli In the structured format, a single template file contains the header, body,

econds to execute Id footer sections, each enclosed within special **<LPHEADER>**, was requested at 04:23:47 of clock **PBODY>**, and **<LPFOOTER>** tags that mark the boundaries of each the Url /mydocuments/styles/style.css, requested by 192.168.1.101, took 49 m clion.

filliseconds to execute. Structured format template files are specified using the <u>tpl</u> parameter. It was requested at 04:23:48 o'clock.

The following is a sample structured format template file:

<LPHEADER>This is my template, for a query containing %FIELDS_NUM % fields, executed by %USERNAME%.</LPHEADER>

Tsonfollowing command parses an IIS log file and creates a text file formatted as queding wether templeter the sted by %c-ip%, took %time-taken

% milliseconds to execute.

Itogasreeu'esfell aC%tinho%@'alackt FROM extend1.log'' -o:TPL -tpl:mytempl at/LBBODY>

There sulting output pyill look like the following example:

</LPFOOTER>

This is my template, for a query containing 32 fields, executed by TestUser.

The Url /default.htm, requested by 192.168.1.102, took 24 milliseconds to exe **Note**: The TPL output format assumes that the character cute.

It was reproted the object of the opening tag for a section, such as

The Url /mydocuments/index.html, requested by 192.168.1.104, took 134 milli seconds to execute.

It was requested at 04:23:47 o'clock.

Templaten Variables ts/styles/style.css, requested by 192.168.1.101, took 49 m

The following table the variables that are available to template files: It was requested at 04:23:48 o'clock.

End of report

Variable	Description	Example Template
%FIELD_n%	Value of the output	First field value: %FIELD_1%

	record field with the specified 1- based index	
%field_name%	Value of the specified output record field	First field value: %SourceName%
%FIELDNAME_n%	Name of the output record field with the specified 1- based index	%FIELDNAME_1% value %FIELD_1%
%FIELDS_NUM%	Number of output record fields	There are %FIELDS_NUM% fields.
%SYSTEM_TIMESTAMP%	Current system date and time, in UTC coordinates	Generated at %SYSTEM_TIMESTAMP
%environment_variable%	Value of the specified environment variable ¹	Generated by %USERNAME%

Notes:

(1): When a variable matches both a field name and an environment variable, the field value is substituted.

TPL Output Format Into-Entity Syntax

<into-entity> ::= <filename> | STDOUT

The <u><into-entity></u> specified in queries using the TPL output format is either:

- A filename;
- The "STDOUT" keyword, which specifies that the output data is to be written to the output stream (the console output).

The default into-entity for queries that do not specify an <u>INTO clause</u> is "STDOUT".

The TPL output format supports the <u>multiplex feature</u>, which can be enabled by specifying '*' wildcards in the into-entity filename. This feature allows output records to be written to different files depending on the values of their fields. For more information on the multiplex feature, see <u>Multiplexing Output Records</u>.

Examples:

INTO MyPage.html

INTO \\COMPUTER01\Reports\report.txt

INTO STDOUT

INTO Reports_*_*\Report*.txt

TPL Output Format Parameters

The TPL output format supports the following parameters:

tpl

Values:	file path
Default:	not specified
Description:	Template file.
Details:	When using raw format template files, this parameter specifies the template file containing the body section. When using structured format template files, this parameter specifies the single template file that contains the header, body, and footer sections. For more information on template files, see <u>Template Files</u> .
Example:	-tpl:MyTemplate.tpl

tplHeader

- Values: file path
- Default: not specified

Description: Template header file.

Details: When using raw format template files, this parameter specifies the template file containing the header section. When using structured format template files, this parameter specifies a raw format template file that overrides the <LPHEADER> section of the structured format template file specified with the "tpl" parameter. For more information on template files, see <u>Template</u> <u>Files</u>. Example: -tplHeader:MyTemplateHeader.tpl

tplFooter

Values:	file path
Default:	not specified
Description:	Template footer file.
Details:	When using raw format template files, this parameter specifies the template file containing the footer section. When using structured format template files, this parameter specifies a raw format template file that overrides the <lpfooter> section of the structured format template file specified with the "tpl" parameter. For more information on template files, see Template Files.</lpfooter>
Example:	-tplFooter:MyTemplateFooter.tpl

noEmptyFile

ON	
	ON

Default: ON

Description: Do not generate empty files.

Details: When a query does not produce output records, the TPL output format does not write a body section, and the resulting output file could be empty. Setting this parameter to "ON" causes the TPL output format to avoid generating an empty file in these situations.

Example: -noEmptyFile:OFF

oCodepage

Values: codepage ID (number)

Default:	0
Description:	Codepage of the output text.
Details:	0 is the system codepage, -1 is UNICODE.
Example:	-oCodepage:1245

fileMode

Values: **0 | 1 | 2**

Default: 1

Description: Action to perform when an output file already exists.

Details: This parameter controls the behavior of the TPL output format when the <u>into-entity</u> specifies directly or indirectly through the "multiplex" feature the name of a file that already exists. The possible values for this parameter are:

. Or existing files are encoded with the even

- 0: existing files are appended with the output;
- 1: existing files are overwritten with the output;
- 2: existing files are left intact, discarding the output.

Example:

-fileMode:0

TPL Output Format Examples

Last 50 Security Events

Create an HTML page containing the most recent 50 events from the Security event log:

LogParser "SELECT TOP 50 TimeGenerated, SourceName, EventID, Messag e INTO Events.html FROM Security" -i:EVT -direction:BW -o:TPL -tpl:HTM LBody.txt -tplHeader:HTMLHeader.txt -tplFooter:HTMLFooter.txt

MSDN BLogs Channel Titles

Display titles of current channels on MSDN BLogs:

LogParser "SELECT title INTO channels.txt FROM http://blogs.msdn.com/M ainFeed.aspx#/rss/channel/item" -i:XML -fMode:Tree -o:TPL -tpl:mytemplate. tpl

TSV Output Format

The TSV output format writes output records as tab-separated or spaceseparated values text.

The output of the TSV output format consists of multiple lines of text, one line for each output record.

Each line contains the values of the output record fields, separated by either a tab character or a space character, depending on the value of the <u>oSeparator</u> parameter.

If enabled through the <u>headers</u> parameter, the first line in the output is a "header" that contains the names of the fields.

The following sample shows the output of the TSV output format when using the default values for its parameters:

EventID	SourceName	EventType	Tim	eGenerated
6009	EventLog4	2004-04-18 1	18:48:	04
6005	EventLog4	2004-04-18 1	18:48:	04
17024 ntit	Service Contro	ol Manager	1	2004-04-18 18:48:27
D7035	Service Contro	ol Manager	4	2004-04-18 18:48:27
	Service Contro	ol Manager	4	2004-04-18 18:48:27
7036	Service Contro	ol Manager	4	2004-04-18 18:48:27
7036	Service Contro	ol Manager	4	2004-04-18 18:48:27
7035	Service Contro	ol Manager	4	2004-04-18 18:48:27
Seegalso	Service Contro	ol Manager	4	2004-04-18 18:48:27
c ⁷ €935Outr	Servicencentro	ol Manager	4	2004-04-18 18:48:27
T2936npu	t Serviset Contro	ol Manager	4	2004-04-18 18:48:27
7035	Service Contro		4	2004-04-18 18:48:27
7036	Service Contro	ol Manager	4	2004-04-18 18:48:27
7035	Service Contro	Yomattagerpo	<u>orațioi</u>	n.2001/110/htsgreserved.
7036	Service Contro	ol Manager	4	2004-04-18 18:48:27
7036	Service Contro	ol Manager	4	2004-04-18 18:48:27
7035	Service Contro	ol Manager	4	2004-04-18 18:48:36
7036	Service Contro	ol Manager	4	2004-04-18 18:51:26
7036	Service Contro	ol Manager	4	2004-04-18 18:51:29
6006	EventLog4	2004-04-18 1	18:51:	37
	-			

TSV Output Format Into-Entity Syntax

<into-entity> ::= <filename> | STDOUT

The <u><into-entity></u> specified in queries using the TSV output format is either:

- A filename;
- The "STDOUT" keyword, which specifies that the output data is to be written to the output stream (the console output).

The default into-entity for queries that do not specify an <u>INTO clause</u> is "STDOUT".

The TSV output format supports the <u>multiplex feature</u>, which can be enabled by specifying '*' wildcards in the into-entity filename. This feature allows output records to be written to different files depending on the values of their fields. For more information on the multiplex feature, see <u>Multiplexing Output Records</u>.

Examples:

INTO report.tsv

INTO \\COMPUTER01\Reports\report.tsv

INTO STDOUT

INTO Reports_*_*\Report*.tsv

TSV Output Format Parameters

The TSV output format supports the following parameters:

headers

Values:	ON OFF AUTO
Default:	AUTO
Description:	Write a header line containing the field names.
Details:	 This parameter controls the header line that is output at the beginning of each file. The possible values for this parameter are: ON: always write the header; OFF: never write the header; AUTO: write the header only when not appending to an existing file.
Example:	-headers:OFF
oSeparator	
Values:	any string space tab
Default:	tab
Description:	Separator between fields.

Details: This parameter controls the separator to be used between field values. The "tab" keyword causes the TSV output format to use a single tab character between the fields, while the "space" keyword causes the TSV output format to use a single space character.

Example: -oSeparator:space

oTsFormat

Values:	timestamp format
Default:	yyyy-MM-dd hh:mm:ss
Description:	Format of timestamp values in the output TSV data.
Details:	This parameter specifies the date and/or time format to use when formatting values of the <u>TIMESTAMP</u> data type. For more information on date and time formats, see <u>Timestamp Format Specifiers</u> .
Example:	-oTsFormat:"MMM dd, yyyy"

oCodepage

Values:	codepage ID (number)
Default:	0
Description:	Codepage of the output text.
Details:	0 is the system codepage, -1 is UNICODE.
Example:	-oCodepage:1245

fileMode

- Values: **0 | 1 | 2**
- Default: 1

Description: Action to perform when an output file already exists.

- Details: This parameter controls the behavior of the TSV output format when the <u>into-entity</u> specifies directly or indirectly through the "multiplex" feature the name of a file that already exists. The possible values for this parameter are:
 - 0: existing files are appended with the output;

- 1: existing files are overwritten with the output;
- 2: existing files are left intact, discarding the output.

Example:

-fileMode:0

TSV Output Format Examples

File Information

Create a TSV file containing information on the files contained in the specified directory:

LogParser "SELECT Path, Name, Size, Attributes INTO Files.tsv FROM C:\T est*.*" -i:FS -o:TSV -recurse:0

Security Events

Retrieve the 10 latest events from the Security event log and write their information to a TSV file for each event ID:

LogParser "SELECT TOP 10 EventID, EventTypeName, Message INTO Even ts_*.tsv FROM Security" -i:EVT -direction:BW -o:TSV

W3C Output Format

The W3C output format writes output records in the W3C Extended Log File Format.

The following example shows a sample output generated by the W3C output format:

#Software: Microsoft Log Parser #Version: 1.0 I#Data10003y10325 14:20:40 P#F201031616161e time s-id s-type s-category E200404-518 18:48:04 6009 4 0 2004-04-18 18:48:04 6005 4 0 2004-04-18 18:48:27 7024 1 0 See also 2004-04-18 18:48:27 7035 4 0 2004-04-18 18:48:27 7035 4 0 2004-04-18 18:48:27 7036 4 0 2004-04-18 18:48:27 7036 4 0 2004-04-18 18:48:27 7036 4 0

W3C Output Format Into-Entity Syntax

<into-entity> ::= <filename> | STDOUT

The <u><into-entity></u> specified in queries using the W3C output format is either:

- A filename;
- The "STDOUT" keyword, which specifies that the output data is to be written to the output stream (the console output).

The default into-entity for queries that do not specify an <u>INTO clause</u> is "STDOUT".

The W3C output format supports the <u>multiplex feature</u>, which can be enabled by specifying '*' wildcards in the into-entity filename. This feature allows output records to be written to different files depending on the values of their fields. For more information on the multiplex feature, see <u>Multiplexing Output Records</u>.

Examples:

INTO report.log

INTO \\COMPUTER01\Reports\report.log

INTO STDOUT

INTO Reports_*_*\Report*.log

W3C Output Format Parameters

The W3C output format supports the following parameters:

rtp

Values:	number of rows
Default:	10
Description:	Rows to print before pausing.
Details:	When writing to STDOUT, the W3C output format displays output records in batches made up of a number of rows equal to the value specified for this parameter. Once a batch of rows has been displayed, the W3C output format prompts the user to press a key to display the next batch of rows. Specifying "-1" for this parameter disables batching altogether.
Example:	-rtp:-1

oDQuotes

L	
Values:	ON OFF
Default:	OFF
Description:	Enclose string values in double-quote characters.
Details:	When this parameter is set to "ON", the W3C output format writes string values with double-quote (") characters around them.
Example:	-oDQuotes:ON

oDirTime

Values: any string

Description: Content of the "#Date" directive header.

Details: The W3C output format uses the value specified for this parameter as the content of the "#Date" directive written to the header of the output file. When a value is not specified, the W3C output format uses the current date and time.

Example: -oDirTime:"1973-05-28 03:02:42"

encodeDelim

- Values: **ON | OFF**
- Default: OFF
- Description: Substitute space characters within field values with plus characters.
- Details: When this parameter is set to "ON", the W3C output format substitutes space characters found in string values with plus (+) characters, in order to generate W3C output that is formatted correctly. When this parameter is set to "OFF", space characters within field values are preserved, potentially generating invalid W3C output.

Example: -encodeDelim:ON

oCodepage

Values:	codepage ID (number)
Default:	0
Description:	Codepage of the output text.
Details:	0 is the system codepage, -1 is UNICODE.
Example:	-oCodepage:1245

fileMode

Values: **0 | 1 | 2**

Default: 1

Description: Action to perform when an output file already exists.

Details: This parameter controls the behavior of the W3C output format when the <u>into-entity</u> specifies directly or indirectly through the "multiplex" feature the name of a file that already exists.

The possible values for this parameter are:

- 0: existing files are appended with the output;
- 1: existing files are overwritten with the output;
- 2: existing files are left intact, discarding the output.

Example:

-fileMode:0

W3C Output Format Examples

Event Log Report

Create a W3C file with information from the System event log:

LogParser "SELECT TO_DATE(TimeGenerated) AS date, TO_TIME(TimeGenerated) AS time, SourceName AS s-source, EventID AS s-event-id, EventCat egory AS s-event-category INTO report.log FROM System" -i:EVT -o:W3C - encodeDelim:ON

XML Output Format

The XML output format writes output records as XML document nodes.

Users can choose between four different structures for the output XML document.

Different structures format the output record fields in different ways, giving users the ability to fine-tune the generated XML for their applications.

The following example command generates an XML document containing fields from the System event log:

LogParser "SELECT TimeGenerated, SourceName, EventID, Message INTO Events.xml FROM System"

The output XML will look like the following example:

<?xml version="1.0" encoding="ISO-10646-UCS-2" standalone="yes" ?>

<!DOCTYPE ROOT[

<!ATTLIST ROOT DATE CREATED CDATA #REQUIRED>

<!ATTLIST.ROOT CREATED_BY CDATA #REQUIRED>

Parameters

<!ELEMENT Message (#PCDATA)>

<!ELEMENT ROW (TimeGenerated, SourceName, EventID, Message)> <!ELEMENT ROOT (ROW*)>

Spee also:

XNOOT PATE CREATED="2004-11-08 16:26:54" CREATED_BY="Micros oft Log Parser V2.2">

<ROW>

<TimeGen@2004 Microsoft Corporation. All rights reserved.

2004-04-18 18:48:04

</TimeGenerated>

<SourceName>

EventLog

</SourceName>

<EventID> 6009 </EventID> <Message> Microsoft (R) Windows (R) 5.01. 2600 Service Pack 1 Uniprocessor Free. </Message> </ROW> <ROW> <TimeGenerated> 2004-04-18 18:48:04 </TimeGenerated> <SourceName> **EventLog** </SourceName> <EventID> 6005 </EventID> <Message> The Event log service was started. </Message> </ROW><ROW> <TimeGenerated> 2004-04-18 18:48:27 </TimeGenerated> <SourceName> Service Control Manager </SourceName> <EventID> 7035 </EventID> <Message> The Network Connections service was successfully sent a start control. </Message> </ROW> </ROOT>

XML Output Format Document Structures

The XML output format generates XML documents that can be structured in four different ways, depending on the value specified for the <u>structure</u> parameter.

Structure 1

When the "structure" parameter is set to "1", the XML output format creates a node named "ROW" for each output record.

This node in turn contains nodes for each field in the output record, named after the field names and with node values containing the field values.

The following example shows an XML document created with structure "1":

<?xml version="1.0" encoding="ISO-10646-UCS-2" standalone="yes" ?>

<!DOCTYPE ROOT[

StructureS2 ROOT DATE_CREATED CDATA #REQUIRED>

Setting the "structure" parameter to "2" causes the XML output format to setting the "structure" parameter to "2" causes the XML output format to generate XML documents that are formatted according to structure "1", and in which field nodes have a "YPE" attribute that specifies the data setting of the corresponding output record field.

The Formula and the standard and the sta

]>

<ROOT DATE "CREATED="2004-11-08 17:36:44" CREATED BY="Micros
{xml version="1.0" encoding="1SO-10646-UCS-2" standalone="yes" ?>

Structure ST ROOT DATE_CREATED CDATA #REQUIRED>

CLATTELIST ROOT CREATED BY CDATA #REOUIRED> When the "structure that ameter is set to "3", the XML output format SELEMENT Time Cenerated (#PCDATA)> creates achoes maneed "ROW" for each output record.

This hode in the contains hodes named "FIELD" for each field in the <!ELEMENT SourceName (#PCDATA)> outplatified gitd Search NENEL DY Prode Datas A#Rode Value Dequal to the field value bath and the field name.

≪ ANTERLINS ¥ EventID TYPE CDATA #REQUIRED>

The fight wing reverse to the structure "3" ATTENT Message TYPE CDATA #REQUIRED>

≪IMeEsteeNT ROW (TimeGenerated, SourceName, EventID, Message)>

< Middledis (H) + Middledis (H

}×12N0NeSTagPE ROOT[

STRUCTURESATIROCREDATED CREDATED OBDABA: #RECREREED_BY="Micros

of RT WIPA TERMOD T'EREATED, BY CDATA #REQUIRED> Setting The "Structure" parameter to "4" causes the XML output format to generate XML occuments that are formatted according to structure "3", **30 In Which FIELD** nodes have an additional "TYPE" attribute that specifies the data type of the corresponding output record field. **30 In Which FIELD** for the corresponding output record field.

The with structure shows an XML document created with structure "A Root contract of the created = "2004-11-08 17:32:41" CREATED_BY="Micros

oft Honor Penkem V2.2">

<

6006LD NAME="Message" TYPE="STRING">

MRH6600 (R) Windows (R) 5.01. 2600 Service Pack 1 Uniprocessor Free.

<###IIDNAME="Message">

TROEVent log service was started.

<REDEAL>D>

```
≪RIEWD NAME="TimeGenerated" TYPE="TIMESTAMP">
```

<2R049074>18 18:48:04

</FIELD>

```
<FIELD NAME="SourceName" TYPE="STRING">
```

EventLog

</FIELD>

```
<FIELD NAME="EventID" TYPE="INTEGER">
```

6005

</FIELD>

```
<FIELD NAME="Message" TYPE="STRING">
```

```
The Event log service was started.
```

```
</FIELD>
```

</ROW>

</ROOT>

XML Output Format Into-Entity Syntax

<into-entity> ::= <filename> | STDOUT

The <u><into-entity></u> specified in queries using the XML output format is either:

- A filename;
- The "STDOUT" keyword, which specifies that the output data is to be written to the output stream (the console output).

The default into-entity for queries that do not specify an <u>INTO clause</u> is "STDOUT".

The XML output format supports the <u>multiplex feature</u>, which can be enabled by specifying '*' wildcards in the into-entity filename. This feature allows output records to be written to different files depending on the values of their fields. For more information on the multiplex feature, see <u>Multiplexing Output Records</u>.

Examples:

INTO report.xml

INTO \\COMPUTER01\Reports\report.xml

INTO STDOUT

INTO Reports_*_*\Report*.xml

XML Output Format Parameters

The XML output format supports the following parameters:

structure

Values:	1 2 3 4
Default:	1
Description:	Structure of the output document.
Details:	For a description of the different structures available, see <u>Document Structures</u> .
Example:	-structure:4

rootName

	Values:	string
	Default:	ROOT
	Description:	Name of the document root node.
	Details:	This parameter allows users to customize the name of the single root node that contains all the other nodes in the output document.
	Example:	-rootName:REPORT
row	Name	

Values:	string
Default:	ROW
Description:	Name of the node containing the output record fields.
Details:	This parameter allows users to customize the name of the node that is generated for each output record.

Example: -rowName:ENTRY

fieldName

Values:	string
Default:	FIELD
Description:	Name of the node containing the output record field valu
Details:	This parameter allows users to customize the name of the node that is generated for each output record field when the "structure" parameter is set to "3" or "4".
Example:	-fieldName:DATA

xslLink

Values:	path to XSL document
---------	----------------------

Default: *not specified*

Description: XSL document to be referenced by the output XML document.

Details: Specifying a value for this parameter causes the XML output format to place a link to the specified XSL stylesh in the header of the output XML document. XSL-enabled XML browsers will follow the specified link and format the output XML document accordingly. The link placed in the document header is formatted as follows:

<?xml-stylesheet type="text/xsl" href="C:\XSL\MyXSL.xsl"

Example: -xslLink:C:\XSL\MyXSL.xsl

schemaType

Values: **0 | 1**

| Default: | 1 |
|--------------|---|
| Description: | Type of inline schema. |
| Details: | When this parameter is set to "1", the output XML
document contains an inline DTD schema.
Setting this parameter to "0" prevents the XML output
format from generating an inline schema. |
| Example: | -schemaType:0 |

compact

| Values: | ON | OFF |
|---------|----|-----|
|---------|----|-----|

Default: OFF

Description: Suppress indentations and extra lines in output.

Details: When this parameter is set to "OFF", the XML output format generates XML documents that are optimized for human readability, indenting nodes according to their depth, and writing nodes on multiple lines. Setting this parameter to "ON" causes the XML output format to write each "ROW" node on a single line withour indentation.

Example: -compact:ON

noEmptyField

- Values: **ON | OFF**
- Default: OFF

Description: Avoid writing empty nodes for NULL field values.

Details: When this parameter is set to "OFF", output record fields having <u>NULL</u> values are rendered as empty nodes. Setting this parameter to "ON" prevents the XML output format from generating a node when the corresponding

output record field has a NULL value.

Example: -noEmptyField:ON

standAlone

| Values: | ON OFF |
|--------------|--|
| Default: | ON |
| Description: | Create a well-formed, stand-alone XML document. |
| Details: | When this parameter is set to "ON", the XML output form
generates well-formed XML documents having an XML
header and a single document root node.
When this parameter is set to "OFF", the XML output
format generates XML text that only contains the output
record nodes, with no XML header and no document roo
node. |
| Example: | -standAlone:OFF |

oCodepage

| Values: | codepage ID (number) |
|--------------|--|
| Default: | 0 |
| Description: | Codepage of the output text. |
| Details: | 0 is the system codepage, -1 is UNICODE. |
| Example: | -oCodepage:1245 |

fileMode

- Values: **0 | 1 | 2**
- Default: 1

Description: Action to perform when an output file already exists.

| Details: | This parameter controls the behavior of the XML output
format when the <u>into-entity</u> specifies directly or indirectly
through the "multiplex" feature the name of a file that |
|----------|--|
| | already exists.
The possible values for this parameter are: |
| | • 0: existing files are appended with the output; |
| | 1: existing files are overwritten with the output; |

• 2: existing files are left intact, discarding the output.

Example: -fileMode:0

XML Output Format Examples

Account Logons

Create an XML document containing logon account names and dates from the Security Event Log messages:

LogParser "SELECT TimeGenerated AS LogonDate, EXTRACT_TOKEN(Str ings, 0, '|') AS Account INTO Report.xml FROM Security WHERE EventID N OT IN (541;542;543) AND EventType = 8 AND EventCategory = 2"

Command-Line Operation

The Log Parser command-line executable is a single, standalone binary file ("LogParser.exe") that can be used from the Windows command-line shell to execute queries and perform other Log Parser tasks. The executable binary does not require any installation; once copied to a computer, it is ready to use.

Tip: If you want to run LogParser.exe from any directory without having to specify the absolute or relative path, you can add the Log Parser directory location to the "PATH" environment variable.

The Log Parser command-line executable works on *commands* supplied by the user. Commands are combinations of *switches*, or arguments, that specify parameters for the task that needs to be executed. The switches used with the Log Parser command-line executable must be entered with a dash character (-) followed by the switch name, as in the following example:

C:\>LogParser -h

Most switches require a user-supplied value; in these cases, the switch name must be followed by a colon character (:) and by the user-supplied value with no intervening spaces, as in the following example:

C:\>LogParser -iCodepage:931

If the user-supplied value contains spaces, the value can be surround by double-quote characters ("), as in the following example:

C:\>LogParser -chartTitle:"Top 20 Pages"

Depending on the switches used in a command, the Log Parser command-line executable can be used in four different *modes* of operation:

• Query Execution Mode: this is the default mode of operation; in this

mode, Log Parser is used to execute queries reading input records from an input format and writing output records to an output format.

- <u>Conversion Mode</u>: in this mode, activated by the "-c" switch, Log Parser is used to execute built-in queries that convert log files between supported log file formats.
- <u>Defaults Override Mode</u>: in this mode, activated by the "saveDefaults" switch, users can override the default behavior of Log Parser by specifying custom default values for the execution parameters.
- <u>Help Mode</u>: in this mode, activated by the "-h" switch, the commandline executable can be used to display to the console window a "quick reference" help on selected topics, such as information on input and output formats, syntax of functions, and syntax of the Log Parser SQL-Like query language.

See also:

Global Switches Reference Commands and Queries

Query Execution Mode

"Query Execution Mode" is the default operational mode of the Log Parser command-line executable.

In this mode, Log Parser is used to execute queries reading input records from an input format and writing output records to an output format.

The general syntax of commands in query execution mode is:

```
LogParser [-i:<input_format>] [<input_format_options>]
[-o:<output_format>] [<output_format_options>]
<SQL query> | file:<query_filename>[?param1=value1+...]
[<global_switches>] [-queryInfo]
```

-i:<input_format>

Specifies the input format for the query.

The "-i:" switch is followed by the name of the selected input format, as in the following example:

C:\>LogParser -i:IISW3C "SELECT * FROM extend1.log"

When an input format is not specified, Log Parser will attempt to select automatically an input format upon inspection of the <fromentity> in the FROM clause. For example, "System" suggests the use of the EVT Input Format, while "ex040302.log" suggests the use of the IISW3C Input Format.

If the <from-entity> does not suggest a specific input format, the <u>TextLine Input Format</u> will be selected by default.

<input_format_options>

Specify values for input format parameters.

These are entered as switches with names matching the input format's parameter names, followed by a colon and by the value for
the parameter, as in the following examples:

C:\>LogParser -i:IISW3C -iCodepage:932 -iCheckpoint:MyCheckpoint.l pc "SELECT * FROM extend1.log"

C:\>LogParser -i:EVT -binaryFormat:ASC "SELECT * FROM System"

Parameter values containing spaces must be enclosed within double-quote characters ("), as in the following example:

C:\>LogParser -i:EVT -stringsSep:"MY SEPARATOR" "SELECT * FRO M System"

For more information on input format parameters, refer to the <u>Input</u> <u>Format Reference</u>.

-o:<output_format>

Specifies the output format for the query. The "**-o:**" switch is followed by the name of the selected output format, as in the following example:

C:\>LogParser -o:CSV "SELECT * FROM System"

When an output format is not specified, Log Parser will attempt to select automatically an output format upon inspection of the <intoentity> in the <u>INTO</u> clause. For example, "chart.gif" suggests the use of the <u>CHART Output Format</u>, while "MyFile.csv" suggests the use of the <u>CSV Output Format</u>.

If the <into-entity> does not suggest a specific output format, or the query does not specify an INTO clause, the <u>NAT Output Format</u> will be selected by default.

<output_format_options>

Specify values for output format parameters. These are entered as switches with names matching the output format's parameter names, followed by a colon and by the value for the parameter, as in the following examples:

C:\>LogParser -o:NAT -rtp:-1 -fileMode:1 "SELECT * FROM System"

C:\>LogParser -o:CSV -tabs:ON "SELECT * FROM System"

Parameter values containing spaces must be enclosed within double-quote characters ("), as in the following example:

C:\>LogParser -o:CHART -chartTitle:"Page Hits per Day" "SELECT date , COUNT(*) FROM extend1.log GROUP BY date" For more information on output format parameters, refer to the <u>Output Format Reference</u>.

<SQL query>

Specifies the text of the Log Parser SQL-Like query. Since a query always contains spaces, the text of the query must be enclosed within double-quote characters ("), as in the following example:

C:\>LogParser "SELECT * FROM System"

Alternatively, a query can be specified through a text file with the "file:" switch, as shown in the next section.

Commands containing both a query text argument and a "file:" switch are considered illegal and return an error.

file:<query_filename>[?param1=value1+...]

Specifies the name of a text file containing a Log Parser SQL-Like query.

The text file specified must contain a valid query in the Log Parser SQL-Like language. Multiple spaces, comments, and new-line characters in the text file are ignored, allowing the query text to be formatted as desired for readability.

The following example shows an example content of a query text file:

SELECT TimeGenerated,

EXTRACT_TOKEN(ResolvedSid, 1, '\\') AS Username -- only the ' Therfollowipgressample shows how the query is executed, assuming that the guery text has here a file named "MyQuery.sql":

*/

USHN6gP#ESOILVETSID(Nip)uAsyBesolvedSid

FROM Security

Query text files can include *parameters*, which are substituted at runtime with user-supplied text or environment variable values. Parameters are user-defined names in the query text enclosed within percent characters (%), such as "%MyParameter%".

When issuing a Log Parser command to execute a query text file containing parameters, users can specify the values of the parameters by appending the question-mark character (?) to the query filename, followed by a list of pairs in the form of "parameter_name=parameter_value", separated by the plus character (+).

For example, the following query contains two parameters:

SELECT EventID

FROM %InputEventLog%

The following example commans executes the query substituting user-supplied values for the parameters:

C:\>LogParser -i:EVT file:Myquery.sql?InputEventLog=System+InputSo urceName=EventLog

If a parameter name or value contains spaces, the name or value must be enclosed within double-quote characters ("), as in the following example:

C:\>LogParser -i:EVT file:Myquery.sql?InputEventLog=System+InputSo urceName="Service Control Manager"

If the value of a query text file parameter is not supplied by the user, Log Parser will search for the parameter name in the current environment variable set. If an environment variable is found matching the parameter name, its value will be substituted for the parameter; otherwise, the parameter name is left as-is in the query text.

The text of the query can also be specified directly as a commandline argument, as shown in the previous section.

Commands containing both a query text argument and a "file:" switch are considered illegal and return an error.

<global_switches>

Global switches control overall behaviors of the command, such as error handling and command statistics verbosity. For more information on global switches, refer to the <u>Global</u> <u>Switches Reference</u>.

-queryInfo

Displays diagnostic information about the command. When "-queryInfo" is specified, the command is not executed, and the following diagnostic information is displayed to the console window:

- The text of the provided query, after being parsed and interpreted by the Log Parser SQL-Like engine core;
- Names of the input and output formats selected;
- Structure of the query output records, including field names and field data types.

This information can be used to troubleshoot a variety of problems, including unexpected query execution results, and query parameter subtitution.

The following example uses the "-queryInfo" switch to display diagnostic information about the specified command:

C:\>LogParser "SELECT TO_UTCTIME(TimeGenerated) AS UTCTime Generated, SourceName FROM System WHERE EventID > 20" -queryIn The output of this command is:

Query:

SELECT TO_UTCTIME([TimeGenerated]) AS UTCTimeGenerated, [S ourceName] FROM System See also Exercised Sector (Sector Sector Secto

Query RefuseAll rights reserved.UTCTimeGenerated (T)SourceName (S)

Conversion Mode

In "Conversion Mode", Log Parser is used to execute built-in queries to convert log files between the following formats:

- <u>BIN</u> to <u>W3C</u>
- <u>IIS</u> to <u>W3C</u>
- BIN to IIS
- IISW3C to IIS

Conversion mode is activated by the "-c" switch.

The general syntax of commands in conversion mode is:

```
LogParser -c -i:<input_format> -o:<output_format> <from_entity>
<into_entity> [ <where_clause> ] [ <input_format_options> ]
[ <output_format_options> ] [ -multiSite[:ON|OFF] ]
[ <global_switches> ] [ -queryInfo ]
```

For more information on log file format conversions, refer to <u>Converting</u> <u>File Formats</u>.

-i:<input_format>

Specifies the input format for the conversion. The "-i:" switch is followed by the name of the selected input format, as in the following example:

```
C:\>LogParser -c -i:IISW3C -o:IIS extend1.log inetsv1.log
```

Differently than <u>Query Execution Mode</u>, the input format specification is a mandatory argument for commands in conversion mode. The specified input format name must be one of the input formats in the table above for which a conversion is supported.

-o:<output_format>

Specifies the output format for the conversion. The "**-o:**" switch is followed by the name of the selected output format, as in the following example:

```
C:\>LogParser -c -i:IISW3C -o:IIS extend1.log inetsv1.log
```

Differently than <u>Query Execution Mode</u>, the output format specification is a mandatory argument for commands in conversion mode.

The specified output format name must be one of the output formats in the table above for which a conversion is supported.

<from_entity>

Specifies the input file(s) to be converted.

This argument must conform to the <<u>from_entity></u> syntax of the selected input format. For information on the syntax and interpretation of the <from_entity> values supported by each input format, refer to the <u>Input Formats Reference</u>.

If the argument contains spaces, it must be enclosed within doublequote characters ("), as in the following example:

C:\>LogParser -c -i:IISW3C -o:IIS "extend1.log;, <1>" inetsv1.log

<into_entity>

Specifies the conversion target output file.

This argument must conform to the <<u>into_entity></u> syntax of the selected output format. For information on the syntax and interpretation of the <into_entity> values supported by each output format, refer to the <u>Output Formats Reference</u>.

If the argument contains spaces, it must be enclosed within doublequote characters ("), as in the following example:

```
C:\>LogParser -c -i:IISW3C -o:IIS extend1.log "C:\My Folder\inetsv1.lo
g"
```

<where_clause>

Specifies an optional <u>WHERE</u> clause to perform filtering on the input format entries.

The following example converts only the IISW3C log file entries that represent successful requests:

C:\>LogParser -c -i:IISW3C -o:IIS extend1.log inetsv1.log "WHERE sc-s tatus BETWEEN 200 AND 399"

<input_format_options>

Specify values for input format parameters.

These are entered as switches with names matching the input format's parameter names, followed by a colon and by the value for the parameter, as in the following example:

C:\>LogParser -c -i:IISW3C -o:IIS extend1.log inetsv1.log -iCodepage:9 32

For more information on input format parameters, refer to the <u>Input</u> <u>Format Reference</u>.

<output_format_options>

Specify values for output format parameters.

These are entered as switches with names matching the output format's parameter names, followed by a colon and by the value for the parameter, as in the following example:

C:\>LogParser -c -i:IISW3C -o:IIS extend1.log inetsv1.log -fileMode:1

For more information on output format parameters, refer to the <u>Output Format Reference</u>.

-multiSite[:ON|OFF]

Specifies that an IIS Central Binary log file is to be converted to multiple log files, one for each IIS Virtual Site. This option is only available when the conversion is from the <u>BIN</u> input format, and when the specified <into-entity> contains one "*" wildcard enabling the <u>Multiplex Ouput Mode</u>. The wildcard will be

replaced with the numeric identifiers of the IIS Virtual Sites that served the requests logged in the central binary log file.

The following example converts a single IIS Central Binary log file to different W3C log files, one for each IIS Virtual Site that served a request logged in the central binary log:

C:\>LogParser -c -i:BIN -o:W3C raw1.ibl C:\NewLogs\W3SVC*\extend 1.log -multiSite:ON

<global_switches>

Global switches control overall behaviors of the command, such as error handling and command statistics verbosity. For more information on global switches, refer to the <u>Global</u> <u>Switches Reference</u>.

-queryInfo

Displays diagnostic information about the conversion command. When "-queryInfo" is specified, the command is not executed, and the following diagnostic information is displayed to the console window:

- The text of the conversion query, after being parsed and interpreted by the Log Parser SQL-Like engine core;
- Names of the input and output formats selected;
- Structure of the query output records, including field names and field data types.

This information can be used to troubleshoot unexpected conversion

results.

The following example uses the "-queryInfo" switch to display diagnostic information about the specified conversion command:

C:\>LogParser -c -i:IISW3C -o:IIS extend1.log inetsv1.log -queryInfo

The output of this command is:

Query:

SELECT [c-ip], [cs-username], TO_DATE(TO_LOCALTIME(TO_TIM ESTAMP([date],

[time]))), TO_TIME(TO_LOCALTIME(TO_TIMESTAMP([date], [time See a]39;

Commarsitenamed bergementernamed [s-ip], [time-taken], [sc-bytes], Global Swhites [Restatus] [sc-win32-status], [cs-method], [cs-uri-stem], Converting i=flee formats

INTO inetsv1.log

FROM extend1 log 2004 Microsoft Corporation. All rights reserved.

Formats selected: Input format: IISW3C (IIS W3C Extended Log Format) Output format: IIS (IIS Log Format)

```
Query fields:

c-ip (S)

cs-username (S)

TO_DATE(TO_LOCALTIME(TO_TIMESTAMP(date, time))) (T)

TO_TIME(TO_LOCALTIME(TO_TIMESTAMP(date, time))) (T)

s-sitename (S)

s-computername (S)

s-computername (S)

s-ip (S)

time-taken (I)

sc-bytes (I)

cs-bytes (I)

sc-status (I)

sc-win32-status (I)
```

cs-method (S) cs-uri-stem (S) cs-uri-query (S)

Defaults Override Mode

In "Defaults Override Mode" users can specify new default values to replace the factory default values of <u>global switches</u>, <u>input format</u> <u>parameters</u>, and <u>output format parameters</u>.

Values are overridden on the computer on which the "saveDefaults" command is executed, and the new values are in effect until they are overridden by a new override command, or until the factory defaults are restored with the "restoreDefaults" command. The new default values also affect the Log Parser scriptable COM components.

Note: For security reasons, properties that are used to specify confidential or sensitive information, such as usernames and passwords, can not be overriden by the "Defaults Override Mode" feature.

The general syntax of commands in defaults override mode is:

| LogParser | <pre>-saveDefaults [-i:<input_format> <input_format_options>]</input_format_options></input_format></pre> | |
|-----------|---|--|
| | [-o: <output_format> <output_format_options>]</output_format_options></output_format> | |
| | [<global_switches>]</global_switches> | |

LogParser -restoreDefaults

-i:<input_format> <input_format_options>

Specifies the input format whose parameters' default values are to be overridden, and the new default values for the selected parameters.

The "-i:" switch is followed by the name of the selected input format, and the new default values are entered as switches with names matching the input format's parameter names, followed by a colon and by the value for the new default, as in the following example:

C:\>LogParser -saveDefaults -i:EVT -binaryFormat:ASC -resolveSIDs:O N

For more information on input format parameters, refer to the Input

Format Reference.

-o:<output_format> <output_format_options>

Specifies the output format whose parameters' default values are to be overridden, and the new default values for the selected parameters.

The "**-o:**" switch is followed by the name of the selected output format, and the new default values are entered as switches with names matching the output format's parameter names, followed by a colon and by the value for the new default, as in the following example:

```
C:\>LogParser -saveDefaults -o:NAT -rtp:-1
```

For more information on output format parameters, refer to the <u>Output Format Reference</u>.

<global_switches>

Specify new default values for global switches.

The following example command overrides the default value of the "stats;" global switch, together with the "rtp" parameter of the <u>NAT</u> output format:

```
C:\>LogParser -saveDefaults -o:NAT -rtp:-1 -stats:OFF
```

For more information on global switches, refer to the <u>Global</u> <u>Switches Reference</u>.

-restoreDefaults

Restores the factory defaults of <u>global switches</u>, <u>input format</u> <u>parameters</u>, and <u>output format parameters</u>. When specified, the "-restoreDefaults" switch must be the only argument of the command, as in the following example:

C:\>LogParser -restoreDefaults

See also:

Command-Line Operation Reference Global Switches Reference

© 2004 Microsoft Corporation. All rights reserved.

Help Mode

"Help Mode", activated with the "**-h**" switch, offers users the possibility to access "quick reference" help topics displayed to the console output. The help topics, selectable through additional command-line arguments, are:

- <u>General Usage</u>
- Query Language Syntax
- Functions Syntax
- Input and Output Formats
- <u>Conversion Mode</u>
- Query Examples

General Usage Help

The Log Parser command-line executable usage help is accessed with the following command:

C:\>LogParser -h

Query Language Syntax Help

The Log Parser SQL-Like language syntax help is accessed with the following command:

C:\>LogParser -h GRAMMAR

Functions Syntax Help

The Log Parser SQL-Like language functions syntax help is accessed

with commands having the following syntax:

LogParser -h FUNC[TIONS] [<function>]

Typing the following command will display the syntax for all the functions available in the Log Parser SQL-Like language:

C:\>LogParser -h FUNCTIONS

Typing a function name following the help command displays the syntax of the selected function only:

C:\>LogParser -h FUNCTIONS SUBSTR

Typing the first few letters of a function name displays the syntax of all the functions whose name starts with the specified letters:

C:\>LogParser -h FUNCTIONS STR

Input and Output Formats Help

Input and output formats help is displayed with commands having the following syntax:

LogParser -h -i:<input_format> [<from_entity>] [<input_format_options>]

LogParser -h -o:<output_format>

For example, the following command displays help on the IISW3C input format:

```
C:\>LogParser -h -i:IISW3C
```

The output of this command gives a detailed overview of the IISW3C

input format, including the syntax of the

<from_entity>, a list of all the supported properties together with their default values, the structure of the records produced by the input format (field names and types), and examples of queries using the input format.

When an input format retrieves field information from the data that needs to be parsed, the help command can include the from-entity from which the field information is to be gathered.

For example, the <u>CSV</u> input format examines the input files to retrieve the names and types of the input record fields that will be exported. A help command aimed at displaying the input record fields exported by the CSV input format when parsing a specific file should include the filename from-entity, as shown in the following example:

C:\>LogParser -h -i:CSV TestLogFile.csv

In addition, since the parameters of some input formats can affect the structure of the input records, help commands can include these parameters to display the varying input record structures. For example, the <u>NETMON</u> input format has a "fMode" parameter that can be used to specify how the input records should be structured. A help command aimed at displaying the input record fields exported by the NETMON input format when the "fMode" parameter is set to "TCPConn" should include this parameter, as shown in the following example:

```
C:\>LogParser -h -i:NETMON -fMode:TCPConn
```

Conversion Mode Help

Conversion mode help is accessed with commands having the following syntax:

```
LogParser -h -c [ -i:<input_format> -o:<output_format> ]
```

The following command displays general conversion mode help,

including the list of available built-in conversion queries:

C:\>LogParser -h -c

The following command displays help on the conversion between the specified log file formats, including the full text of the built-in query that performs the conversion:

C:\>LogParser -h -c -i:BIN -o:W3C

Query Examples Help

Examples of queries and commands can be displayed with the following command:

C:\>LogParser -h EXAMPLES

See also:

Command-Line Operation Reference

© 2004 Microsoft Corporation. All rights reserved.

Global Switches

Global switches control overall behaviors of a command, and they are used with most of the Log Parser command-line executable operational modes.

The global switches are:

-e:<max_errors>

-iw[:ON|OFF]

-stats[:ON|OFF]

-q[:ON|OFF]

-e:<max_errors>

Specifies a maximum number of parse errors to collect internally before aborting the execution of the command.

The default value for this global switch is **-1**, which is a special value causing the SQL engine to ignore *all* parse errors and report only the total number of parse errors encountered during the execution of the command.

The following example command sets the maximum number of parse errors to 100:

```
C:\>LogParser "SELECT Message FROM System" -e:100
```

For more information on parse errors and the "-e" switch, see <u>Errors</u>, <u>Parse Errors</u>, and <u>Warnings</u>.

-iw[:ON|OFF]

Specifies whether or not warnings should be ignored.

The default value is "OFF", meaning that run time warnings will not be ignored and will trigger an interactive prompt to the user. Specifying "ON", on the other hand, disables the interactive prompt, and run time warnings will be ignored and their total count will be reported when the command execution has completed.

The following example command executes a query ignoring run time warnings:

C:\>LogParser "SELECT Message FROM System" -iw:ON

For more information on warnings and the "-iw" switch, see <u>Errors</u>, <u>Parse Errors</u>, and <u>Warnings</u>.

-stats[:ON|OFF]

Specifies whether or not command execution statistics should be displayed when the command execution has completed.

The default value is "ON", causing command execution statistics to be always displayed. Specifying "OFF" prevents the statistics from being displayed.

The following example command executes a query preventing the statistics from being displayed:

C:\>LogParser "SELECT COUNT(*) FROM System" -stats:OFF

-q[:ON|OFF]

Enables or disables "quiet mode".

When "quiet mode" is enabled, the console output of a command contains only the output records, suppressing any additional information. For this reason, the console output of a command executed in "quiet mode" is suitable to be redirected to a text file. Enabling "quiet mode" disables the display of <u>parse errors</u>, <u>warnings</u>, and statistics. In addition, if the selected output format is the <u>NAT</u> output format, its "rtp" and "headers" parameters are automatically set as follows:

- -rtp:-1
- -headers:OFF

As an example, the output of following command shows the extra information and the NAT output format headers that are normally displayed to the console:

C:\>LogParser "SELECT COUNT(*) FROM System"

```
COUNT(ALL *)
```

In this example, enabling "quiet mode" suppresses the headers displayed by the NAT output format and the query execution statistics, and the output would look like the following:

Statistics:

G:\>LogParser "SELECT COUNT(*) FROM System" -q:ON

E9diments processed: 6913

Elements output: 1

Execution time: 0.13 seconds

See also:

<u>Command-Line Operation Reference</u> <u>Errors, Parse Errors, and Warnings</u>

© 2004 Microsoft Corporation. All rights reserved.

COM API

The Log Parser scriptable COM components architecture is made up of the following objects:

- <u>LogQuery object</u>: this object is the main COM object in the Log Parser scriptable COM components architecture; it exposes methods to execute SQL-Like queries and provides access to global parameters controlling the execution of a query.
- <u>LogRecordSet object</u>: this object is an enumerator of LogRecord objects; it allows an application to navigate through the output records of a query.
- LogRecord object: this object represents a single query output record, and it exposes methods that can be used to retrieve individual field values from the output record.
- <u>Input Format objects</u>: these objects provide programmatic access to the input formats supported by Log Parser; each input format object exposes properties having the same name as the parameters of the corresponding Log Parser input format.
- <u>Output Format objects</u>: these objects provide programmatic access to the output formats supported by Log Parser; each output format object exposes properties having the same name as the parameters of the corresponding Log Parser output format.

See also:

Log Parser COM API Overview C# Example

© 2004 Microsoft Corporation. All rights reserved.

LogQuery Object

The **LogQuery** object exposes the main API methods that execute a SQL-Like query and provides access to global parameters controlling the execution of a query.

The object is instantiated with the "**MSUtil.LogQuery**" Progld. The class name of the <u>.NET COM wrapper</u> for this object is "**Interop.MSUtil.LogQueryClassClass**".

Methods

| Execute | Executes a query and returns a <u>LogRecordSet</u> object that can be used to navigate through the query output records. |
|--------------|--|
| ExecuteBatch | Executes a query and writes the query output records to an output format. |

Properties

| errorMessages | Returns a collection of the <u>error</u> , <u>parse</u>
<u>error</u> , and <u>warning</u> messages that
occurred during the execution of a query. |
|---------------------|---|
| inputUnitsProcessed | Returns the total number of input records processed during the execution of a query. |
| lastError | Returns -1 if <u>errors</u> , <u>parse errors</u> , or <u>warnings</u> occurred during the execution |

| | of the query; 0 otherwise. |
|----------------------|--|
| maxParseErrors | Sets and gets the maximum number of parse errors that can occur during the execution of a query before aborting the query execution. |
| outputUnitsProcessed | Returns the total number of output
records sent to an output format during
the execution of a query. |
| <u>versionMaj</u> | Returns the "major" component of the version of the Log Parser scriptable COM components. |
| versionMin | Returns the "minor" component of the version of the Log Parser scriptable COM components. |

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

VBScript example:

Dim oLogQuery

Set oLogQuery = CreateObject("MSUtil.LogQuery")

See also:

LogRecordSet Object Input Format Objects Output Format Objects Log Parser COM API Overview C# Example

© 2004 Microsoft Corporation. All rights reserved.

Execute Method

Executes a query and returns a <u>LogRecordSet</u> object that can be used to navigate through the query output records.

Script Syntax

objRecordSet = objLogQuery.Execute(strQuery [, objInputFormat]);

Parameters

strQuery

A string containing the text of the SQL-Like query to be executed.

objInputFormat

Either an <u>Input Format object</u> or a <u>Custom Input Format Plugin</u> object.

If this parameter is not specified, or is *null*, Log Parser will attempt to select automatically an input format upon inspection of the <fromentity> in the <u>FROM</u> clause of the specified query.

Return Value

A <u>LogRecordSet</u> object, which can be used to navigate through the query output records.

Remarks

• If the query execution encounters <u>errors</u>, an exception is thrown containing the error message and code, and the query execution is aborted.

In this case, the <u>lastError</u> property of the LogQuery object is set to **-1**,

and the collection of strings returned by the <u>errorMessages</u> property contains the error message.

- If the query execution encounters <u>parse errors</u> or <u>warnings</u>, the query executes successfully, and the method returns a LogRecordSet object. In this case, the <u>lastError</u> property of the LogQuery object is set to -1, and the collection of strings returned by the <u>errorMessages</u> property contains the parse error messages and/or warning messages.
- A successful execution of the Execute method does not necessarily mean that the query execution has completed.
 Depending on the query structure, navigating the query output records with the LogRecordSet object can cause the query to further process new input records, which could in turn generate additional errors, parse errors, or warnings. See the LogRecordSet Object Reference for more information.
- The specified query can not contain an <u>INTO</u> clause.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

V/BScriptlepathple:at object

var oIISW3CInputFormat = new ActiveXObject("MSUtil.LogQuery.IIS
MacJappyEquery.i");

Dim oIISW3CInputFormat Dim oIISW3CInputFormat Dim offerent Concepts text Dim offerent Concepts and the second s

LogPinestr Client P Execute query and receive a LogRecordSet ExecuteBatch Method LogRecordSet Coject LogRecordSet Coject Input Format Objects Log Parser Press Part Content of Conten

Set In Sweet of the CreateObject("MSUtil.LogQuery.IISW3CInp

C# (Example')

// Get a record

'Creater que exond the march of the courd Set get Racional ()All rights reserved. strQuery = "SELECT c-ip FROM <1> WHERE cs-uri-stem LIKE '%hitc ount.#s6Et first field value var strClientIp = oRecord.getValue(0); ' Execute query and receive a LogRecordSet Set oRetrond Siele vallog Query. Execute (strQuery, oIISW3CInputFormat) WScript.Echo("Client IP Address: " + strClientIp); ' Visit all records DO WHAId Van NO Log Record Set tate next record oRecordSet.moveNext(); ' Get a record } Set oRecord = oRecordSet.getRecord // Close LogRecordSet oRec'ordes finstosie(); value strClientIp = oRecord.getValue(0)' Print field value WScript.Echo "Client IP Address: " & strClientIp

' Advance LogRecordSet to next record oRecordSet.moveNext

LOOP

' Close RecordSet oRecordSet.close

ExecuteBatch Method

Executes a query and writes the output records to an output format.

Script Syntax

bResult = objLogQuery.ExecuteBatch(strQuery [, objInputFormat [, objOutput
Format]]);

Parameters

strQuery

A string containing the text of the SQL-Like query to be executed.

objInputFormat

Either an Input Format object or a <u>Custom Input Format Plugin</u> object.

If this parameter is not specified, or is *null*, Log Parser will attempt to select automatically an input format upon inspection of the <fromentity> in the <u>FROM</u> clause of the specified query.

objOutputFormat

An Output Format object.

If this parameter is not specified, or is *null*, Log Parser will attempt to select automatically an output format upon inspection of the <intoentity> in the <u>INTO</u> clause of the specified query.

Return Value

A boolean value. Returns *TRUE* if the query executed with <u>parse errors</u> or <u>warnings</u>; *FALSE* if the query executed without any parse error nor warning.

Remarks

- If the query execution encounters <u>errors</u>, an exception is thrown containing the error message and code, and the query execution is aborted.
 In this case, the <u>lastError</u> property of the LogQuery object is set to -1, and the collection of strings returned by the <u>errorMessages</u> property contains the error message.
- If the query execution encounters <u>parse errors</u> or <u>warnings</u>, the query executes successfully, and the method returns *TRUE*.
 In this case, the <u>lastError</u> property of the LogQuery object is set to -1, and the collection of strings returned by the <u>errorMessages</u> property contains the parse error messages and/or warning messages.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

V/BScraiptleparthplerat object

var oEVTInputFormat = new ActiveXObject("MSUtil.LogQuery.EventL

pshipit5gcmet");

DEN DE VITRO HIM BANK BUT BW";

Dim oCSVOutputFormat

协航驾行Quitput Format object

SeevalsOSVOutputFormat = new ActiveXObject("MSUtil.LogQuery.CSVO

Log View Format direction = "Application Popup";

' Create Output Format object // Execute query 6EtogQ&V@HpecEtoBatch(CheQueObjecE(VIN6plttEbog@ueGySUSOUCpupEtoF orand;") oCSVOutputFormat.tabs = TRUE

' Create query text strQuery = "SELECT TimeGenerated, EventID INTO C:\output.csv FRO M System" strQuery = strQuery & "WHERE SourceName = 'Application Popup'"

' Execute query

oLogQuery.ExecuteBatch strQuery, oEVTInputFormat, oCSVOutputFor mat

errorMessages Property

Returns a collection of strings containing the messages of <u>errors</u>, <u>parse</u> <u>errors</u>, or <u>warnings</u> encountered while executing a query with the <u>Execute</u> or <u>ExecuteBatch</u> methods.

Read-only property.

Script Syntax

value = objLogQuery.errorMessages;

Return Value

A collection of Strings containing error messages.

Remarks

• The object returned by the errorMessages property implements a single read-only _NewEnum property. The _NewEnum property retrieves an IEnumVARIANT interface on an object that can be used to enumerate the collection.

The _NewEnum property is hidden within scripting languages (JScript and VBScript). Applications written in the JScript language handle objects implementing the _NewEnum property as **Enumerator** objects or with the **for...in** statement, while applications written in the VBScript language handle objects implementing the _NewEnum property with the **For Each...Next** statement.

• If you want to retrieve <u>parse error</u> messages, make sure that the <u>maxParseErrors</u> property of the <u>LogQuery</u> object is set to a value different than **-1**. If the value of this property is **-1** (the default value), the parse error messages will be discarded, and the errorMessages collection will contain a single message stating the total number of parse errors occurred.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

VBSlokiptuexalmplerse error messages are collected

```
oLogQuery.maxParseErrors = 100;
```

Dim oLogQuery

Different Oglervy text

var strQuery = "SELECT sc-bytes INTO C:\\output.csv FROM ex040528.

Seff"@LogQuery = CreateObject("MSUtil.LogQuery")

See also:

Logo Level and Logo Level parse error messages are collected Locolgeguery has purchase goory);

C# Example //Createrrotexpccurred

新QLOBQUESFIELET SC: bytes INTO C:\output.csv FROM ex040528.log" { <u>© 2004 Microsoft Corporation. All rights reserved</u>.

```
' EWeSgript Ferbo("Errors occurred!");
```

```
oLogQuery.ExecuteBatch strQuery
```

var oMessages = new Enumerator(oLogQuery.errorMessages);

```
' Cfree(k 操作 ( ); oMessages.moveNext())
```

```
If oLogQuery.lastError <> 0 Then
```

```
WScript.Echo("Error message: " + oMessages.item());
```

WScript.Echo "Errors occurred!"

For Each strMessage In oLogQuery.errorMessages

```
else WScript.Echo "Error Message: " + strMessage
```

```
{ Next
```

```
WScript.Echo("Executed successfully!");
```

Èlse

WScript.Echo "Executed succesfully!"

End If

inputUnitsProcessed Property

Returns the total number of input records processed by a query executed with the <u>ExecuteBatch</u> method.

Read-only property.

Script Syntax

value = objLogQuery.inputUnitsProcessed;

Return Value

An integer value containing the total number of input records processed by the last query executed with the <u>ExecuteBatch</u> method.

Remarks

• When a query is executed with the <u>Execute</u> method, this property returns zero. In these cases, use the <u>inputUnitsProcessed</u> property of the <u>LogRecordSet</u> object.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

VBScriptqexample:

```
var strQuery = "SELECT TimeGenerated, EventID INTO C:\\output.csv
```

DiQuerQuery "WHERE SourceName = 'Application Popup''';

See Set song Query CreateObject("MSUtil.LogQuery")

oLogQuery,ExecuteBatch(strQuery); LogQuery Object

<u>ExecuteBatch</u> Method <u>Subject voist Info</u>tementeredde prontes an TO C:\output.csv FRO <u>outputChilsPlocessed Ploperty</u> <u>MSystem Chol "Input Records Processed: " + oLogQuery.inputUnitsProc</u> Log Parser COM API Overview DE Conversion Population Population strQuery & "WHERE SourceName = 'Application Popup'"

'Execute query © 2004 Microsoft Corporation. All rights reserved. oLogQuery.ExecuteBatch strQuery

' Display total number of input records processed

WScript.Echo "Input Records Processed: " & oLogQuery.inputUnitsProce ssed
lastError Property

Returns **-1** if the <u>Execute</u> or <u>ExecuteBatch</u> methods encountered <u>errors</u>, <u>parse errors</u>, or <u>warnings</u>; **0** otherwise.

Read-only property.

Script Syntax

value = objLogQuery.lastError;

Return Value

An integer value containing **-1** if the <u>Execute</u> or <u>ExecuteBatch</u> methods encountered <u>errors</u>, <u>parse errors</u>, or <u>warnings</u>; **0** otherwise.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

VBScriptqexample:

var strQuery = "SELECT TimeGenerated, EventID INTO C:\\output.csv **DRA VLSystery**"; **DRA VLSystery** "WHERE SourceName = 'Application Popup'';

Śድ እና ይሳት የንድ CreateObject("MSUtil.LogQuery") See blso uery.ExecuteBatch(strQuery);

Loc Greate Output text Loc Weiser Configure Generated, EventID INTO C:\output.csv FRO C# V. gl. og Guery.lastError != 0) strQuery = strQuery & "WHERE SourceName = 'Application Popup'" WScript.Echo("Errors occurred!"); } © 2004 Microsoft Corporation. All rights reserved.

eEscecute query

&LogQuery.ExecuteBatch strQuery

WScript.Echo("Executed successfully!");

Check if errors occurred

If oLogQuery.lastError <> 0 Then

WScript.Echo "Errors occurred!"

Else

WScript.Echo "Executed succesfully!" End If

maxParseErrors Property

Sets or gets the maximum number of <u>parse errors</u> that can occur during the execution of a query before aborting the query execution.

Read/write property.

Script Syntax

objLogQuery.maxParseErrors = value;

value = objLogQuery.maxParseErrors;

Argument/Return Value

An integer value specifying the maximum number of <u>parse errors</u> that can occur during the execution of a query before aborting the query execution.

A value of **-1** specifies that all parse errors should be ignored.

Default Value

-1

Remarks

• This property is analogous to the "-e" <u>global switch</u> available with the Log Parser command-line executable.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

VBSgQptrgxampleseErrors = 10;

Dim oLogQuery Set oLogQuery = CreateObject("MSUtil.LogQuery")

oLogQuery.maxParseErrors = 10 **See also:**

LogQuery Object Log Parser COM API Overview C# Example

© 2004 Microsoft Corporation. All rights reserved.

outputUnitsProcessed Property

Returns the total number of output records sent to an output format by a query executed with the <u>ExecuteBatch</u> method.

Read-only property.

Script Syntax

value = objLogQuery.outputUnitsProcessed;

Return Value

An integer value containing the total number of output records sent to an output format by the last query executed with the <u>ExecuteBatch</u> method.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

VBScriptqexample:

var strQuery = "SELECT TimeGenerated, EventID INTO C:\\output.csv ERAMLSyquer'y: ERAMERE SourceName = 'Application Popup''';

ሄድ እቴርዓቴዊ ባዛዮ ሃ = CreateObject("MSUtil.LogQuery") See bls Query.ExecuteBatch(strQuery);

Log Greate Output text Execute Barch Method input Greate Written: "+ oLogQuery.outputUnitsProc inputUnitsProcessed Property Loger Can Darry Conversion Popup'' C# Example

- ' Execute query

oLogQuerg. Execute Readon of Comporation. All rights reserved.

' Display total number of output records generated

WScript.Echo "Output Records Written: " & oLogQuery.outputUnitsProc essed

versionMaj Property versionMin Property

Return the *major* and *minor* components of the version of the Log Parser scriptable COM components currently being used.

Read-only properties.

Script Syntax

value = objLogQuery.versionMaj;

value = objLogQuery.versionMin;

Return Values

Integer values containing the *major* and *minor* components of the version of the Log Parser scriptable COM components currently being used.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

```
VBScriptFexa(hipleg Parser Version " + oLogQuery.versionMaj + "." + oL ogQuery.versionMin );
```

Dim oLogQuery

Set oLogQuery = CreateObject("MSUtil.LogQuery")

WScript.Echo "Log Parser Version " & oLogQuery.versionMaj & "." & o

Sed also ery.versionMin

LogQuery Object Log Parser COM API Overview C# Example

© 2004 Microsoft Corporation. All rights reserved.

LogRecordSet Object

The **LogRecordSet** object is returned by the <u>Execute</u> method of the <u>LogQuery</u> object, and it exposes methods that can be used to navigate through the output records of a query.

The **LogRecordSet** object is an enumerator of **LogRecord** objects.

The interface name of the <u>.NET COM wrapper</u> for this object is "Interop.MSUtil.ILogRecordset".

| atEnd | Returns a Boolean value indicating if the enumerator is at the end of the collection. |
|-----------------------|---|
| close | Releases the enumeration and all the associated resources. |
| <u>getColumnCount</u> | Returns the number of fields in the query output records. |
| <u>getColumnName</u> | Returns the name of a field in the query output records. |
| <u>getColumnType</u> | Returns the data type of a field in the query output records. |
| getRecord | Returns the current LogRecord object in the enumeration. |
| moveNext | Advances the enumerator to the next
LogRecord in the enumeration. |

Methods

Properties

| errorMessages | Returns a collection of the <u>error</u> , <u>parse</u>
<u>error</u> , and <u>warning</u> messages that
occurred during the last invocation of the
<u>moveNext</u> method. |
|---------------------|---|
| inputUnitsProcessed | Returns the total number of input records processed during the execution of a query. |
| lastError | Returns -1 if <u>errors</u> , <u>parse errors</u> , or
<u>warnings</u> occurred during the last
invocation of the <u>moveNext</u> method; 0
otherwise. |
| INTEGER_TYPE | Returns the value of the constant representing the INTEGER data type. |
| NULL_TYPE | Returns the value of the constant representing the NULL data type. |
| REAL_TYPE | Returns the value of the constant representing the REAL data type. |
| STRING_TYPE | Returns the value of the constant representing the STRING data type. |
| TIMESTAMP_TYPE | Returns the value of the constant representing the TIMESTAMP data type. |

Examples

JScript example:

```
var oLogQuery = new ActiveXObject("MSUtil.LogQuery");
var oLogRecordSet = oLogQuery.Execute( "SELECT * FROM System" )
VBScript example:
```

Dim oLogQuery Dim oLogRecordSet

```
Set oLogQuery = CreateObject("MSUtil.LogQuery")
SetSalsDogRecordSet = oLogQuery.Execute( "SELECT * FROM System" )
```

LogQuery Object LogRecord Object Log Parser COM API Overview C# Example

© 2004 Microsoft Corporation. All rights reserved.

atEnd Method

Returns a Boolean value indicating if the enumerator is at the end of the collection.

Script Syntax

value = objRecordSet.atEnd();

Return Value

A Boolean value set to *TRUE* if there are no more <u>LogRecord</u> objects to enumerate; *FALSE* otherwise.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

V/BScriptlepathple:at object

var oIISW3CInputFormat = new ActiveXObject("MSUtil.LogQuery.IIS
biacLappyEquertat");

Dim oIISW3CInputFormat

b mean Query text

ይዩኩፍቴሬ የሚያስት የ Select c-ip FROM <1> WHERE cs-uri-stem LIKE '% Sechalsonteesp'd;

LogRecord Object LogRecord Object LogRecord Object LogRecord Object LogSet Regenered and receive a LogRecordSet LogSet Regenered and receive a LogRecordSet LogSet Regenered and receive a LogRecordSet LogRecord Object C# Example //Creste Infert Polemat object Schild(SecordSet FatEnd(), CreateObject("MSUtil LogOuery.IISW3CInp (utFormat")

// Get a record

'CreateropRegutest⊨ oRecordSet.getRecord();

strQuery = "SELECT c-ip FROM <1> WHERE cs-uri-stem LIKE '%hitc ount.#s**6**'et first field value

```
var strClientIp = oRecord.getValue( 0 );
```

' Execute query and receive a LogRecordSet

```
Set oR Proint Siel & vallog Query. Execute (strQuery, oIISW3CInputFormat)
```

```
WScript.Echo( "Client IP Address: " + strClientIp );
```

' Visit all records

DO WHAIdstante @ LogReccord Sout and End xt record

oRecordSet.moveNext();

} 'Get a record

```
Set oRecord = oRecordSet.getRecord
```

```
// Close LogRecordSet
```

```
oRec'ordes fusiosie(); value
```

```
strClientIp = oRecord.getValue ( 0 )
```

' Print field value WScript.Echo "Client IP Address: " & strClientIp

' Advance LogRecordSet to next record oRecordSet.moveNext

LOOP

' Close RecordSet oRecordSet.close

close Method

Releases the enumeration and all the associated resources.

Script Syntax

objRecordSet.close();

Return Value

None.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

VBScriptlepartTplerat object var oIISW3CInputFormat = new ActiveXObject("MSUtil.LogQuery.IIS biaConputGounat"); Dim oIISW3CInputFormat biaGoungtoury text panetoRecordse"SELECT c-ip FROM <1> WHERE cs-uri-stem LIKE '% SectionSectorse"SELECT c-ip FROM <1> WHERE cs-uri-stem LIKE '% SectionSectorse" LogRecordSetSetSetEctorse LogRecordSetEctorse LogRecordSetEctorse LogRecordSetEctorse VCreateInperference "CreateObject("MSUtil LogOuery, IISW3CInputFormat"); C# Example "CreateObject("MSUtil LogOuery, IISW3CInputFormat"); // Get a record

```
'CreateropRegutedt⊨ oRecordSet.getRecord();
```

strQuery = "SELECT c-ip FROM <1> WHERE cs-uri-stem LIKE '%hitc ount.#s6'et first field value

```
var strClientIp = oRecord.getValue( 0 );
```

' Execute query and receive a LogRecordSet

```
Set oR Proind Sield vallog Query. Execute (strQuery, oIISW3CInputFormat)
WScript. Echo("Client IP Address: " + strClientIp );
```

' Visit all records

DO WHAIdstantel@EogReccordSettatenetxt record

oRecordSet.moveNext();

} 'Get a record

Set oRecord = oRecordSet.getRecord

// Close LogRecordSet

```
oRec'ordes diustofie(); value
```

```
strClientIp = oRecord.getValue ( 0 )
```

' Print field value WScript.Echo "Client IP Address: " & strClientIp

' Advance LogRecordSet to next record oRecordSet.moveNext

LOOP

' Close RecordSet oRecordSet.close

getColumnCount Method

Returns the number of fields in the query output records.

Script Syntax

value = objRecordSet.getColumnCount();

Return Value

An integer value containing the number of fields in the query output records.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

VBScriptqexample:

var strQuery = "SELECT * FROM System";

Dim oLogQuery

炒莊xerute gutgyt and receive a LogRecordSet yaroRecordSet = oLogQuery.Execute(strQuery);

'Execute query and receive a LogRecordSet Set/dRecordSet = oLogQuery.Execute (strQuery) switch(oRecordSet.getColumnType(f)) ' D(splay field names and types

For fcase bRoRoadSetSENgetGEIRmFK/DEnt()-1{ WScript.Echo("Field Type: INTEGER"); ' Field Name break: WScript.Echo "Field Name: " & oRecordSet.getColumnName(f) 'FicelsetyBecordSet.REAL TYPE: { Select Case oRecordSet.get@@aniptTkycleof()"Field Type: REAL"); break; WScript.Echo "Field Type: Case oRecordSet.INTEGER TYPE INTEGER" WScript.Echo "Field Type: R Casse of Reccondisett STIBAIN GTYPPE: WScript.Echo("Field Type: STRING"); EAL" Case oRecordSet.STRINbreat&PE WScript.Echo "Field Type: S TRING" } Case oRecordSet.TIMESTAMP_TYPE WScript.Echo "Field Typ e: TIMESTAMO d'dSet.TIMESTAMP TYPE: { Case oRecordSet.NULL_WVSetEpt.EchoWScienter.Expres 'IHMESTAR' ULL" break; **End Select** } Nextcase oRecordSet.NULL_TYPE: WScript.Echo("Field Type: NULL"); ' Close LogRecordSet break; oRecordSet.close() } } } // Close LogRecordSet

oRecordSet.close();

getColumnName Method

Returns the name of a field in the query output records.

Script Syntax

value = objRecordSet.getColumnName(index);

Parameters

index

The 0-based index of the field in the query output records. The index must be less than the number of fields returned by the <u>getColumnCount</u> method.

Return Value

A string value containing the name of the output record field at the specified position.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

VBScriptqexample:

var strQuery = "SELECT * FROM System";

Dim oLogQuery

/ኦሑዮጵጵዮጵጵዮጵዮ and receive a LogRecordSet ይጫኩ የRecordSet = oLogQuery.Execute(strQuery); **See**/**abso:** and types Sant(vlange)uefyoReroads@bjæt(('öMisUnfilolung(Qufery')) LogRecordSet Object og¹ Parser COM API Overview C//efielduNiantext c# Example strQuarypt.ESEb(E'CFTetdERadM: SysteRe'cordSet.getColumnName(f)); ' Executed graph and receive a LogRecord Set SetwReb(mBetordSetgQtleoyLExeType(f)t)Query) { 'Displaye for doord for divitie for TYPE: { For f = 0 To oRecordSet.getCollinationEat()(1"Field Type: INTEGER"); break: ' Field Name } WScript.Echo "Field Name: " & oRecordSet.getColumnName(f) case oRecordSet.REAL_TYPE: { ' Field type WScript.Echo("Field Type: REAL"); Select Case oRecordSet.get**bio**kmnType(f) Case oRecordSet.INTEGER TYPE WScript.Echo "Field Type: { INTEGERRECORDSet.STRING TYPE: Case oRecordSet.REAL_WSPEpt.EchoWScienter.SchowScienter.Sc EAL" break; Case oRecordSet.STRING_TYPE WScript.Echo "Field Type: S TRING" Casse of Reccondisett TITINGESTRANGP TIMPE: WScript.Echo "Field Typ WScript.Echo("Field Type: TIMESTAMP" e: TIMESTAMP" Case oRecordSet.NULL_TYPE WScript.Echo "Field Type: N): ULL" break; } End Select case oRecordSet.NULL TYPE: Next WScript.Echo("Field Type: NULL"); break; ' Close LogRecordSet } oRecordSet.close() }

// Close LogRecordSet
oRecordSet.close();

getColumnType Method

Returns the type of a field in the query output records.

Script Syntax

value = objRecordSet.getColumnType(index);

Parameters

index

The 0-based index of the field in the query output records. The index must be less than the number of fields returned by the <u>getColumnCount</u> method.

Return Value

An integer value containing the type of the output record field at the specified position. This value is one of the constants returned by the <u>INTEGER_TYPE</u>, <u>REAL_TYPE</u>, <u>STRING_TYPE</u>, <u>TIMESTAMP_TYPE</u>, and <u>NULL_TYPE</u>

properties.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

VBScriptqexample:

var strQuery = "SELECT * FROM System";

HExecute query and receive a LogRecordSet binoBacordSet = oLogQuery.Execute(strQuery); Dim f // Display field names and types LogRecordSet Object Overview: FROM: System FROM: System C# \$#\$\$ ' Execute duely and receive a LogRecordSet. Set offection of the second set of the s ' Display field names and types R_TYPE: For f = 0 To oRecordSet.getColumnCount()_1"Field Type: INTEGER"); break: ł ' Field Name WScript.Echo "Field Name: " & oRecordSet.getColumnName(f) case oRecordSet.REAL_TYPE: { WScript.Echo("Field Type: REAL"); ' Field type Select Case oRecordSet.getCollinnType(f) Case oRecordSet.INTEGER_TYPE WScript.Echo "Field Type: INTEGER Record Set. STRING_TYPE: WSPEPT.Echow Script.Expe: SFIRING"): R Case oRecordSet.REAL break; EAL" Case oRecordSet.STRING TYPE WScript.Echo "Field Type: S TRING" Ease ORECOIDSEL THMESTAMP TYPE 近初始_工学生 「WScript.Echo "Field Typ WScript.Echo("Field Type: TIMESTAMP" e: TIMESTAMP"); Case oRecordSet.NULL TYPE WScript.Echo "Field Type: N break; ULL" } End Select case oRecordSet.NULL TYPE: WScript.Echo("Field Type: NULL"); Next break; } ' Close LogRecordSet

oR}cordSet.close()

}

// Close LogRecordSet
oRecordSet.close();

getRecord Method

Returns the current LogRecord object in the enumeration.

Script Syntax

objRecord = objRecordSet.getRecord();

Return Value

The current LogRecord object in the enumeration.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

VBScriptlexarfiplerat object var oIISW3CInputFormat = new ActiveXObject("MSUtil.LogQuery.IIS ViaConputGourgat"); Dim oIISW3CInputFormat variation variation

// Get a record

```
'CreateropRegutedt⊨ oRecordSet.getRecord();
```

strQuery = "SELECT c-ip FROM <1> WHERE cs-uri-stem LIKE '%hitc ount.#s6'et first field value

```
var strClientIp = oRecord.getValue( 0 );
```

' Execute query and receive a LogRecordSet

```
Set oR Proind Sield vallog Query. Execute (strQuery, oIISW3CInputFormat)
WScript. Echo("Client IP Address: " + strClientIp );
```

' Visit all records

DO WHAIdstantel@EogReccordSettatenetxt record

oRecordSet.moveNext();

} 'Get a record

Set oRecord = oRecordSet.getRecord

// Close LogRecordSet

```
oRec'ordes diustofie(); value
```

```
strClientIp = oRecord.getValue ( 0 )
```

' Print field value WScript.Echo "Client IP Address: " & strClientIp

' Advance LogRecordSet to next record oRecordSet.moveNext

LOOP

' Close RecordSet oRecordSet.close

moveNext Method

Advances the enumerator to the next LogRecord in the enumeration.

Script Syntax

objRecordSet.moveNext();

Return Value

None.

Remarks

- Depending on the query structure, calling the moveNext method can cause the query to further process new input records, which could in turn generate additional <u>errors</u>, <u>parse errors</u>, or <u>warnings</u>.
- If the moveNext method encounters <u>errors</u>, an exception is thrown containing the error message and code, and further processing is aborted.

In this case, the <u>lastError</u> property of the LogRecordSet object is set to **-1**, and the collection of strings returned by the <u>errorMessages</u> property contains the error message.

 If the moveNext method encounters parse errors or warnings, the enumerator is advanced successfully, and the lastError property of the LogRecordSet object is set to -1. In this case, the collection of strings returned by the errorMessages property contains the parse error messages and/or warning messages.

Examples

JScript example:

```
var oLogQuery = new ActiveXObject("MSUtil.LogQuery");
```

VBScriptIexample: at object

var oIISW3CInputFormat = new ActiveXObject("MSUtil.LogQuery.IIS

Macanputopernat");

Dim oIISW3CInputFormat

b Great Oglery text

Sectional Section Sect

LogRecord Object LogRecord Object Log Ser BEERER ALE CLEAR DER FOR AUSTRIC STORE BUS W3CInputFormat); C# Example //Cleateelhpeterdemat object

schild(15083cordSeFatEnd(), CreateObject("MSUtil LogOuery IISW3CInp ttFormat")

// Get a record

'Creater of Record Set.getRecord();

strQuery = "SELECT c-ip FROM <1> WHERE cs-uri-stem LIKE '%hitc ount. #s Get first field value

```
var strClientIp = oRecord.getValue( 0 );
```

```
'Execute query and receive a LogRecordSet
```

Set of Record Sield Value Query. Execute (strQuery, oIISW3CInputFormat) WScript.Echo("Client IP Address: " + strClientIp);

' Visit all records

```
DO WANDER OF OF A CONSECTION AND A CONSECTION OF A CONSECTION 
                                                                                                                                               oRecordSet.moveNext();
```

} ' Get a record

Set oRecord = oRecordSet.getRecord

```
// Close LogRecordSet
```

oRecordset slose(); value

strClientIp = oRecord.getValue(0)

' Print field value WScript.Echo "Client IP Address: " & strClientIp

' Advance LogRecordSet to next record

oRecordSet.moveNext

LOOP

' Close RecordSet oRecordSet.close

errorMessages Property

Returns a collection of strings containing the messages of <u>errors</u>, <u>parse</u> <u>errors</u>, or <u>warnings</u> that occurred during the last invocation of the <u>moveNext</u> method.

Read-only property.

Script Syntax

value = objLogRecordSet.errorMessages;

Return Value

A collection of Strings containing error messages.

Remarks

• The object returned by the errorMessages property implements a single read-only _NewEnum property. The _NewEnum property retrieves an IEnumVARIANT interface on an object that can be used to enumerate the collection.

The _NewEnum property is hidden within scripting languages (JScript and VBScript). Applications written in the JScript language handle objects implementing the _NewEnum property as **Enumerator** objects or with the **for...in** statement, while applications written in the VBScript language handle objects implementing the _NewEnum property with the **For Each...Next** statement.

If you want to retrieve parse error messages, make sure that the maxParseErrors property of the LogQuery object is set to a value different than -1. If the value of this property is -1 (the default value), the parse error messages will be discarded, and the errorMessages collection will contain a single message stating the total number of parse errors occurred.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

VBS/okiptuexalmplerse error messages are collected

oLogQuery.maxParseErrors = 100;

Dim oLogQuery

DGreatestweet Formes object

panolicylacylinputFormat = new ActiveXObject("MSUtil.LogQuery.IIS MacLinutFormat");

See Also Record

Lookecold Set Object

```
//Makeutnewery and ecceivenelson Recordsellected
who green the and the second constraints reserved.
who green the strain of the second term of terms of the second term of terms of terms of term of terms of terms
```

//GebackTifpenr prosranaurigelct

```
ទី៤១៦ជនQV3CXhptEFសាក់ដ ម)CreateObject("MSUtil.LogQuery.IISW3CInp
utFormat")
```

```
WScript.Echo("Errors occurred!");
```

' Create query text

```
strQueryoMssagesTeqpFronerator(wHpsQueryerrordHebsters%hitc
ount.fgf(;;'!oMessages.atEnd(); oMessages.moveNext())
```

```
'Execute Script Enchot Europh Dosnaeord Sol Messages.item());
```

```
Set oRecordSet = oLogQuery.Execute ( strQuery, oIISW3CInputFormat )
}
```

```
' Check if errors occurred
```

```
        It VisitgQutty9ftdstError <> 0 Then while( !oRecordSet.atEnd() )
```

```
{ WScript.Echo "Errors occurred!"
```

```
// Get a record
  For & ack etch/dessegetonolSeggeterycentol()) Messages
     WScript.Echo "Error Message: " + strMessage
  NextGet first field value
     var strClientIp = oRecord.getValue( 0 );
End If
    // Print field value
     WScript.Echo( "Client IP Address: " + strClientIp );
' Visit all records
DO WHAId Van NO Log Record Set tate next record
     oRecordSet.moveNext();
     ' Get a record
     Set bedcifrer for Recond Set. get Record
     if(oRecordSet.lastError != 0)
     '{Get first field value
     str@lemtipt.EclReecEndoget&aduered!'));
     ' Primt 61/desvages = new Enumerator( oRecordSet.errorMessages );
     WSom(ptdMeos'agetentHRdA)ddoesses'sagestufibieeNpxt())
       {
     'AdvaviserIpugRenco("deSectomessageco"rel oMessages.item());
     oRecordSet.moveNext
    }
}
    ' Check if errors occurred
     If oRecordSet.lastError <> 0 Then
// Close LogRecordSet
oRecord/Seccipse()ho "Errors occurred!"
       For Each strMessage In oRecordSet.errorMessages
          WScript.Echo "Error Message: " + strMessage
       Next
     End If
LOOP
' Close RecordSet
oRecordSet.close
```

inputUnitsProcessed Property

Returns the total number of input records processed so far by a query executed with the <u>Execute</u> method.

Read-only property.

Script Syntax

value = objLogRecordSet.inputUnitsProcessed;

Return Value

An integer value containing the total number of input records processed so far by the query that returned the <u>LogRecordSet</u> object.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

V/BSceriptqexample:

var strQuery = "SELECT * FROM System";

```
Sec Signal Query de CreateObject ("MSUtil.LogQuery")

while (!oRecordSet.atEnd())

Log CordSet Object

Log Firster Context ()

Log Firster Context ()

C# ErQuery System ()

C# ErQuery System ()

C# ErQuery ()

C# ErQ
```

WScript.Echo("Input Records Processed: " + oRecordSet.inputUnits 2004 Microsoft Corporation. All rights reserved. PExcessed givery and receive a LogRecordSet

```
Set oRecordSet = oLogQuery.Execute( strQuery )
```

// Get a record

'VisitvalloRecords = oRecordSet.getRecord();

DO WHILE NOT oRecordSet.atEnd

// Advance LogRecordSet to next record

bRisplad/Setumbered/feixt());t records processed so far

} WScript.Echo "Input Records Processed: " & oRecordSet.inputUnits Processed

// Display total number of input records processed

```
WScrifteEalro(diffbtal Input Records Processed: " + oRecordSet.inputUnit sProcessedRecord = oRecordSet.getRecord
```

// Clds&dvægRedægBæcordSet to next record oRecoRd&æcdbæte()pveNext

LOOP

' Display total number of input records processed WScript.Echo "Total Input Records Processed: " & oRecordSet.inputUnit sProcessed

' Close RecordSet oRecordSet.close

lastError Property

Returns **-1** if <u>errors</u>, <u>parse errors</u>, or <u>warnings</u> occurred during the last invocation of the <u>moveNext</u> method; **0** otherwise.

Read-only property.

Script Syntax

value = objRecordSet.lastError;

Return Value

An integer value containing **-1** if the last <u>moveNext</u> method invocation encountered <u>errors</u>, <u>parse errors</u>, or <u>warnings</u>; **0** otherwise.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

// Execute query and receive a LogRecordSet

```
vMakenmedBard # abio greatly Escaperation (Alexied IIS W3 Elwed tFormat);
```

```
oLogQuery.maxParseErrors = 100
```

```
// Check if errors occurred
```

```
ifCoeatgQueeny.FastEntoobject)
```

```
$et oIISW3CInputFormat = CreateObject("MSUtil.LogQuery.IISW3CInp
utForWiStchipt.Echo("Errors occurred!");
```

'Creatarquidigs stages = new Enumerator(oLogQuery.errorMessages); strQufeny(;=!o'lStlebstagets.attpnEtR)OMMetssatgets.frReferences.ext(-);tem LIKE '%hitc ount.asp'''

```
WScript.Echo("Error message: " + oMessages.item());
'Exe&ute query and receive a LogRecordSet
```

```
$et oRecordSet = oLogQuery.Execute ( strQuery, oIISW3CInputFormat )
```

```
//CVhischailflereconsdoccurred
```

While(gQReryotalsterraterrat()))Then

```
{
```

```
WS/ccipt.Æchcor#Errors occurred!"
```

```
var oRecord = oRecordSet.getRecord();
```

```
For Each strMessage In oLogQuery.errorMessages
```

```
WSetifitsEdheldEndoneMessage: "+ strMessage
```

```
Newar strClientIp = oRecord.getValue( 0 );
```

```
End Iff Print field value
```

WScript.Echo("Client IP Address: " + strClientIp);

```
'Visit/aAldeancesLogRecordSet to next record
DO WRHAEdS@TrueReeNerd@et.atEnd
```

```
//Ceheckeifoedrors occurred
fet@RRecordSet.laRt&GoodSet.@etRecord
{
    Get SinsipfiElchv@lterrors occurred!");
    strClientIp = oRecord.getValue ( 0 )
        var oMessages = new Enumerator( oRecordSet.errorMessages );
    ' Pfin(;flelMivaslages.atEnd(); oMessages.moveNext())
W$cript.Echo "Client IP Address: " & strClientIp
```

WScript.Echo("Error message: " + oMessages.item()); ' A}dvance LogRecordSet to next record JoRecordSet.moveNext

}

'Check if errors occurred // CldseoRegRedSerdEestError <> 0 Then oRecordSet.close(); WScript.Echo "Errors occurred!"

> For Each strMessage In oRecordSet.errorMessages WScript.Echo "Error Message: " + strMessage Next

End If LOOP

' Close RecordSet oRecordSet.close
INTEGER_TYPE Property

The constant value returned by the <u>getColumnType</u> method to indicate that an output record field contains values of the <u>INTEGER</u> data type.

Read-only property.

Script Syntax

value = objRecordSet.INTEGER_TYPE;

Return Value

An integer value containing the constant that represents the **INTEGER** data type.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

V/BSceriptqexample:

var strQuery = "SELECT * FROM System";

Dim oLogQuery **拉麻 e Rute of Wey and receive a LogRecordSet**

<u>ይ</u>ያኩρRecordSet = oLogQuery.Execute(strQuery);

Sec absorbag (ield national defended by Ref("MSUtil.LogQuery") for(var f=0; f<oRecordSet.getColumnCount(); f++) NULL TYPE Property REAC reate puery destruy STRUCK defended by Ref ("Field Name: " + oRecordSet.getColumnName(f)); TIMESTAMP. TYPE Property Lod Redelinde Geter bandt receive a LogRecordSet Locsesment of the the second s C# Example 'Displaye for doord fost divitie for TYPE: { For f = 0 to zprecovd Set get Configuration Field Type: HNTE GER"); break; ' Field Name } WScript.Echo "Field Name: " & oRecordSet.getColumnName(f) case oRecordSet.REAL_TYPE: { ' Field type WScript.Echo("Field Type: REAL"); Select Case oRecordSet.get@wakmnType(f) Case oRecordSet.INTEGER_TYPE WScript.Echo "Field Type: INTEGERRECORDSet.STRING TYPE: { Case oRecordSet.REAL_WSPEEpt.EchoWSFieldt.Explor. SFIELUNEybe: R EAL" break; Case oRecordSet.STRING_TYPE WScript.Echo "Field Type: S TRING" Casse of Reccondisett TITINGESTRANDE TIMPE: WScript.Echo "Field Typ WScript.Echo("Field Type: TIMESTAMP" e: TIMESTAMP" WScript.Echo "Field Type: N Case oRecordSet.NULL TYPE); ULL" break; } End Select case oRecordSet.NULL TYPE: { WScript.Echo("Field Type: NULL"); Next break; ' Close LogRecordSet } oRecordSet.close() } // Close LogRecordSet oRecordSet.close();

NULL_TYPE Property

The constant value returned by the <u>getColumnType</u> method to indicate that an output record field contains values of the <u>NULL</u> data type.

Read-only property.

Script Syntax

value = objRecordSet.NULL_TYPE;

Return Value

An integer value containing the constant that represents the $\underline{\text{NULL}}$ data type.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

V/BSceriptqexample:

var strQuery = "SELECT * FROM System";

Dim oLogQuery Dim oL

Sec Dipolog Queld national deferring of the contract of the co

Lod Redelinde Geter bandt receive a LogRecordSet Locsesment of the the second s C# Example 'Displaye for doord fost divitie for TYPE: { For f = 0 To zprecovd Set get Configuration Field Type: HNTE GER"); break; ' Field Name } WScript.Echo "Field Name: " & oRecordSet.getColumnName(f) case oRecordSet.REAL_TYPE: { ' Field type WScript.Echo("Field Type: REAL"); Select Case oRecordSet.get@wakmnType(f) Case oRecordSet.INTEGER_TYPE WScript.Echo "Field Type: INTEGERRECORDSet.STRING TYPE: { Case oRecordSet.REAL_WSPEEpt.EchoWSFieldt.Explor. SFIELUNEybe: R EAL" break; Case oRecordSet.STRING_TYPE WScript.Echo "Field Type: S TRING" Casse of Reccondisett TITINGESTRANDE TIMPE: WScript.Echo "Field Typ WScript.Echo("Field Type: TIMESTAMP" e: TIMESTAMP" WScript.Echo "Field Type: N Case oRecordSet.NULL TYPE); ULL" break; } End Select case oRecordSet.NULL TYPE: { WScript.Echo("Field Type: NULL"); Next break; ' Close LogRecordSet } oRecordSet.close() } // Close LogRecordSet oRecordSet.close();

REAL_TYPE Property

The constant value returned by the <u>getColumnType</u> method to indicate that an output record field contains values of the <u>REAL</u> data type.

Read-only property.

Script Syntax

value = objRecordSet.REAL_TYPE;

Return Value

An integer value containing the constant that represents the $\ensuremath{\underline{\mathsf{REAL}}}$ data type.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

VBScriptqexample:

var strQuery = "SELECT * FROM System";

Dim oLogQuery Dim oRecordSet panpRecordSet = oLogQuery.Execute(strQuery);

Sec Dipolog Queld national Control of the Control o

Lod Redelinde Geter bandt receive a LogRecordSet Locsesment of the the second s C# Example 'Displaye for doord fost divitie for TYPE: { For f = 0 To zprecovd Set get Configuration Field Type: HNTE GER"); break; ' Field Name } WScript.Echo "Field Name: " & oRecordSet.getColumnName(f) case oRecordSet.REAL_TYPE: { ' Field type WScript.Echo("Field Type: REAL"); Select Case oRecordSet.get@wakmnType(f) Case oRecordSet.INTEGER_TYPE WScript.Echo "Field Type: INTEGERRECORDSet.STRING TYPE: { Case oRecordSet.REAL_WSPEEpt.EchoWSFieldt.Explor. SFIELUNEybe: R EAL" break; Case oRecordSet.STRING_TYPE WScript.Echo "Field Type: S TRING" Casse of Reccondisett TITINGESTRANDE TIMPE: WScript.Echo "Field Typ WScript.Echo("Field Type: TIMESTAMP" e: TIMESTAMP" WScript.Echo "Field Type: N Case oRecordSet.NULL TYPE); ULL" break; } End Select case oRecordSet.NULL TYPE: { WScript.Echo("Field Type: NULL"); Next break; ' Close LogRecordSet } oRecordSet.close() } // Close LogRecordSet oRecordSet.close();

STRING_TYPE Property

The constant value returned by the <u>getColumnType</u> method to indicate that an output record field contains values of the <u>STRING</u> data type.

Read-only property.

Script Syntax

value = objRecordSet.STRING_TYPE;

Return Value

An integer value containing the constant that represents the <u>STRING</u> data type.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

V/BSceriptqexample:

var strQuery = "SELECT * FROM System";

Dim oLogQuery

<u>ይ</u>ጫከ የRecordSet = oLogQuery.Execute(strQuery);

Sec Dipolog Query") for(var f=0: f < oRecordSet.getColumnCount(); f++) NULC repto property NULC repto property REAL OF SECT * FROM System" TIMESTAMP. FCD Field Name: " + oRecordSet.getColumnName(f)); Lod Redelinde Geter bandt receive a LogRecordSet Locsesment of the the second s C# Example 'Displaye for doord fost divitie for TYPE: { For f = 0 To zprecovd Set get Configuration Field Type: HNTE GER"); break; ' Field Name } WScript.Echo "Field Name: " & oRecordSet.getColumnName(f) case oRecordSet.REAL_TYPE: { ' Field type WScript.Echo("Field Type: REAL"); Select Case oRecordSet.get@wakmnType(f) Case oRecordSet.INTEGER_TYPE WScript.Echo "Field Type: INTEGERRECORDSet.STRING TYPE: { Case oRecordSet.REAL_WSPEEpt.EchoWSFieldt.Explor. SFIELUNEybe: R EAL" break; Case oRecordSet.STRING_TYPE WScript.Echo "Field Type: S TRING" Casse of Reccondisett TITINGESTRANDE TIMPE: WScript.Echo "Field Typ WScript.Echo("Field Type: TIMESTAMP" e: TIMESTAMP" WScript.Echo "Field Type: N Case oRecordSet.NULL TYPE); ULL" break; } End Select case oRecordSet.NULL TYPE: { WScript.Echo("Field Type: NULL"); Next break; ' Close LogRecordSet } oRecordSet.close() } // Close LogRecordSet oRecordSet.close();

TIMESTAMP_TYPE Property

The constant value returned by the <u>getColumnType</u> method to indicate that an output record field contains values of the <u>TIMESTAMP</u> data type.

Read-only property.

Script Syntax

value = objRecordSet.TIMESTAMP_TYPE;

Return Value

An integer value containing the constant that represents the **<u>TIMESTAMP</u>** data type.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

V/BSceriptqexample:

var strQuery = "SELECT * FROM System";

Dim oLogQuery Dim oLogQuery Dim oLogQuery and receive a LogRecordSet painoRecordSet = oLogQuery.Execute(strQuery);

Sec Dipolog Queld national of the Constraint of

Lod Redelinde Geter bandt receive a LogRecordSet Locsesment of the the second s C# Example 'Displaye for doord fost divitie for TYPE: { For f = 0 To zprecovd Set get Configuration Field Type: HNTE GER"); break; ' Field Name } WScript.Echo "Field Name: " & oRecordSet.getColumnName(f) case oRecordSet.REAL_TYPE: { ' Field type WScript.Echo("Field Type: REAL"); Select Case oRecordSet.get@wakmnType(f) Case oRecordSet.INTEGER_TYPE WScript.Echo "Field Type: INTEGERRECORDSet.STRING TYPE: { Case oRecordSet.REAL_WSPEEpt.EchoWSFieldt.Explor. SFIELUNEybe: R EAL" break; Case oRecordSet.STRING_TYPE WScript.Echo "Field Type: S TRING" Casse of Reccondisett TITINGESTRANDE TIMPE: WScript.Echo "Field Typ WScript.Echo("Field Type: TIMESTAMP" e: TIMESTAMP" WScript.Echo "Field Type: N Case oRecordSet.NULL TYPE); ULL" break; } End Select case oRecordSet.NULL TYPE: { WScript.Echo("Field Type: NULL"); Next break; ' Close LogRecordSet } oRecordSet.close() } // Close LogRecordSet oRecordSet.close();

LogRecord Object

The **LogRecord** object represents a single query output record, and it exposes methods that can be used to retrieve individual field values from the output record.

The **LogRecord** object is returned by the <u>getRecord</u> method of the <u>LogRecordSet</u> object.

The interface name of the <u>.NET COM wrapper</u> for this object is "Interop.MSUtil.ILogRecord".

| <u>getValue</u> | Returns the value of a field in the output record. |
|-------------------|---|
| <u>getValueEx</u> | Returns the value of a field in the output record. |
| <u>isNull</u> | Returns a Boolean value indicating if an output record field is NULL. |
| toNativeString | Returns a field or the whole output record as a string value. |

Methods

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

V/BScriptlepartiple:at object

var oIISW3CInputFormat = new ActiveXObject("MSUtil.LogQuery.IIS

Macongeograt");

Dim oIISW3CInputFormat

b fareat Query text

ይዝከፍቴ Reesty d'S el SELECT c-ip FROM <1> WHERE cs-uri-stem LIKE '% Sed integration of the set of t

LogRiecotristentboject Log Parser COM APLOVERVIEW C# Kar eRegular = Of the Object (*Msteristrogerver) SW3CInputFormat);

```
//CYisitealhactordamat.object.
Schild(1908) 2002 Microsoft Corporation. All rights reserved.
Schild(1908) Schröder attaile CreateObject("MSUtil.LogQuery.IISW3CInp
utFormat")
     // Get a record
'Creater or RecordSet.getRecord();
strQuery = "SELECT c-ip FROM <1> WHERE cs-uri-stem LIKE '%hitc
ount. ds Get first field value
     var strClientIp = oRecord.getValue( 0 );
' Execute query and receive a LogRecordSet
Set of Record Sield Value Query. Execute (strQuery, oIISW3CInputFormat)
     WScript.Echo( "Client IP Address: " + strClientIp );
' Visit all records
DO WHARdyance to BRecord Set to Break record
     oRecordSet.moveNext();
}
     ' Get a record
     Set oRecord = oRecordSet.getRecord
// Close LogRecordSet
oRecords fight value
     strClientIp = oRecord.getValue(0)
     ' Print field value
     WScript.Echo "Client IP Address: " & strClientIp
     ' Advance LogRecordSet to next record
     oRecordSet.moveNext
```

LOOP

' Close RecordSet oRecordSet.close

getValue Method

Returns the value of the field at the specified position in the record.

Script Syntax

value = objRecord.getValue(index);

value = objRecord.getValue(fieldName);

Parameters

index

An integer containing the 0-based index of the field in the query output records. The index must be less than the number of fields returned by the <u>getColumnCount</u> method of the <u>LogRecordSet</u> object.

fieldName

A string containing the name of the field in the query output records.

Return Value

The value of the specified field.

The value is returned as a **VARIANT** (i.e. a scripting variable) whose type depends on the <u>data type</u> of the field. The following table shows the VARIANT type returned and the corresponding scripting types for each of the Log Parser data types:

| Field Type | VARIANT
Type | JScript Type | VBScript
Type |
|------------|-----------------|--------------|------------------|
| INTEGER | VT_I4 | number | Long |

| REAL | VT_R8 | number | Double |
|-----------|---------|-------------------|--------|
| STRING | VT_BSTR | string | String |
| TIMESTAMP | VT_DATE | date (VB
date) | Date |
| NULL | VT_NULL | null object | Null |

Remarks

- Some scripting languages might not handle correctly the *null* value returned by the getValue method when the field at the specified location is <u>NULL</u>. In these cases, call the <u>isNull</u> method before the getValue method to test the field for NULL values.
- Although the Log Parser INTEGER Data Type is a 64-bit value, the getValue method returns INTEGER values as 32-bit integers, since scripting languages do not handle correctly 64-bit integer values. This means that truncation might occur when values are larger than the maximum 32-bit value.

In these cases, if a low-level programming language is being used (e.g. C++), applications can call the <u>getValueEx</u> method to retrieve INTEGER values as 64-bit values.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

VBScriptqexample:

```
var strQuery = "SELECT TimeGenerated, SourceName, EventID, Messag
DFR @Mgystem";
```

```
Dim oRecordSet
   Diffx structergry and receive a LogRecordSet
SedDahsdecordSet = oLogQuery.Execute( strQuery );
Dim val.
           gRecords overview
gRecords et attend () ("MSUtil.LogQuery")
C#
   'CreateGeneryetexatd
   strQuenyoRespiceCorporation, All rights reserved.
   ROM System"
       // Display record information
   'Exet Viscouper Fichnel' televine General Record Ser Record.get Value ("Time Genera
   feet")RecordSet = oLogQuery.Execute( strQuery )
        WScript.Echo( "SourceName : " + oRecord.getValue(1) );
   'VisiWaScreptaEdsho( "EventID : " + oRecord.getValue(2) );
   DO WHERE NOT is Red 6Bd Set.atEnd
        {
        'Get ScreptEcho( "Message
                                   : " + oRecord.getValue(3) );
        $et oRecord = oRecordSet.getRecord
        else
        '{Display record information
        WSvfSptiEtcEoHd(i#Messageted:: "<&udReoprd.getValue("TimeGenera
   ted")}
        WScript.Echo "SourceName : " & oRecord.getValue(1)
       WSdripnEchogReentdBet to hextoRecordd.getValue(2)
        bRecordsetistNuke(Sext(Balse Then
   }
          WScript.Echo "Message : " & oRecord.getValue(3)
        Else
   // CloseW6gRpc&rclSet'Message
                                    : <null>"
   oRecEndSHt.close();
        ' Advance LogRecordSet to next record
        oRecordSet.moveNext
```

LOOP

' Close RecordSet

oRecordSet.close

getValueEx Method

Returns the value of the field at the specified position in the record. The value returned by the getValueEx method is intended for low-level programming languages and is not suitable for consumption by scripting languages.

C++ Syntax

HRESULT getValueEx(IN VARIANT *pindexOrName, OUT VARIANT *pVa l);

Parameters

pindexOrName

A VT_I4 or VT_BSTR VARIANT containing either the 0-based index of the field in the query output records, or the name of the field in the query output records.

The index must be less than the number of fields returned by the <u>getColumnCount</u> method of the <u>LogRecordSet</u> object.

Return Value

The value of the specified field.

The value is returned as a **VARIANT** whose type depends on the <u>data</u> <u>type</u> of the field. The following table shows the VARIANT type returned for each of the Log Parser data types:

| Field Type | VARIANT
Type | Description |
|------------|-----------------|----------------|
| INTEGER | VT_I8 | 64-bit integer |

| REAL | VT_R8 | 64-bit floating-point number |
|---------------|---------|--|
| <u>STRING</u> | VT_BSTR | String |
| TIMESTAMP | VT_I8 | 64-bit integer representing the
number of 100-nanosecond intervals
since January 1, year 0 |
| NULL | VT_NULL | VT_NULL VARIANT |

Remarks

• The getValueEx method returns 64-bit integer values that are not handled correctly by scripting languages, For this reason, the method is intended for use by low-level, non-scripting languages, such as C++. If you are developing an application using scripting languages, consider using the <u>getValue</u> method instead.

See also:

LogRecord Object getValue Method Log Parser COM API Overview C# Example

© 2004 Microsoft Corporation. All rights reserved.

isNull Method

Returns a Boolean value indicating if an output record field is NULL.

Script Syntax

value = objRecord.isNull(index);

value = objRecord.isNull(fieldName);

Parameters

index

An integer containing the 0-based index of the field in the query output records. The index must be less than the number of fields returned by the <u>getColumnCount</u> method of the <u>LogRecordSet</u> object.

fieldName

A string containing the name of the field in the query output records.

Return Value

A Boolean value indicating if the specified output record field is <u>NULL</u>.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

VBScriptqexample:

bim oBogQuery SELECT TimeGenerated, SourceName, EventID, Messag

Dim strQuery

HExecute query and receive a LogRecordSet

See B cordSet = oLogQuery.Execute(strQuery);

LogRecord Object Log Sel Story Appreate Object ("MSUtil.LogQuery") while 'o RecordSet.atEnd()) C# Example ¹Create query text strQuery a second T TimeGenerated, SourceName, EventID, Message, Data FROM System ' Execute Riery and information Record Set Set oRecord Set = oLogQuery.Execute(strQuery d.getValue("TimeGenera ted")); 'Visit all records the ("SourceName" : " + oRecord.getValue(1)); DO WINLE NOT ORECOIDSet.atEnd⁺ oRecord.getValue(2)); if(!oRecord.isNull(3)) ⁴Get a record Set ORecord = ORecord Set.getRecord else Display record information WScript.Echo "TimeGenerated: " & oRecord.getValue("TimeGenera WScript.Echo("Message : <null>"); ted") WScript.Echo "SourceName : " & oRecord.getValue(1) WScript.Echo "EventID :: " & oRecord.getValue(2) if(Record is Null(5) P#alle Then { WScript.Echo "Message WScript.Echo("Data : " & oRecord.getValue(3) : " + oRecord.getValue(4)); [}] WScript.Echo "Message : <null>" EISE If If or the second of the second [}] WScript.Echo "Data : " & oRecord.getValue(4) Else Advance LogRecordSet to next record eRecordSet.moveNext();

}
'Advance LogRecordSet to next record
// ClosRelcogeRsecondSetNext
oRecordSet.close();
LOOP

' Close RecordSet oRecordSet.close

toNativeString Method

Returns a field or the whole output record as a string value.

Script Syntax

value = objRecord.toNativeString(index);

value = objRecord.toNativeString(separator);

Parameters

index

An integer containing the 0-based index of a field in the query output records. The index must be less than the number of fields returned by the <u>getColumnCount</u> method of the <u>LogRecordSet</u> object.

separator

A string containing the separator to be used between the fields of the record.

Return Value

If a field index is used as argument, the method returns the specified field formatted to a string according to the input format string representation of the data type. For example, if the input format used parses timestamps formatted as 'yyyy-MM-dd hh:mm:ss', then the method formats TIMESTAMP values using the same format.

If a string separator is used as argument, the method returns the concatenation of all the record fields formatted to a string, separated by the specified separator.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

VBScriptoexample:

var strQuery = "SELECT TimeGenerated, SourceName, EventID, Messag

DFROM System";

Dim oRecordSet

扮攝來發現現ery and receive a LogRecordSet

γβηρRecordSet = oLogQuery.Execute(strQuery);

Seenalsoal

LogRecord Object

Log Parse Repric Sciented Biact ("MSUtil.LogQuery")

C# Example Create query quert

strQueary oR content of the state of the sta ROM System

// Display record information

'ExeWiscripting chain "Econocenergiede or deseRecord.toNativeString(0)); Set of Set of Set chor by the set of the strong of the strong of the set of t

' Visit/andvancesLogRecordSet to next record DO WRAGE de Tronver de la tend }

' Get a record // Closet Long Refered Set ecord Set.get Record oRecordSet.close();

> ' Display record information WScript.Echo "TimeGenerated: " & oRecord.toNativeString(0) WScript.Echo "Whole Record: " & oRecord.toNativeString(", ")

' Advance LogRecordSet to next record oRecordSet.moveNext

LOOP

' Close RecordSet oRecordSet.close

Input Format Objects

Input Format objects provide programmatic access to the <u>input formats</u> supported by Log Parser.

Input Format objects are instantiated with the ProgId and the <u>.NET COM</u> <u>wrapper</u> class names specified in the following table:

| Input
Format | ProgId | .NET COM |
|-----------------|---------------------------------------|-----------|
| ADS | MSUtil.LogQuery.ADSInputFormat | COMADSI |
| BIN | MSUtil.LogQuery.IISBINInputFormat | COMIISBI |
| CSV | MSUtil.LogQuery.CSVInputFormat | COMCSVIr |
| ETW | MSUtil.LogQuery.ETWInputFormat | COMETWI |
| EVT | MSUtil.LogQuery.EventLogInputFormat | COMEventl |
| <u>FS</u> | MSUtil.LogQuery.FileSystemInputFormat | COMFileSy |
| HTTPERR | MSUtil.LogQuery.HttpErrorInputFormat | COMHttpEı |
| IIS | MSUtil.LogQuery.IISIISInputFormat | COMIISIIS |
| IISODBC | MSUtil.LogQuery.IISODBCInputFormat | COMIISOD |
| IISW3C | MSUtil.LogQuery.IISW3CInputFormat | COMIISW3 |
| NCSA | MSUtil.LogQuery.IISNCSAInputFormat | COMIISNC |
| NETMON | MSUtil.LogQuery.NetMonInputFormat | COMNetMc |
| REG | MSUtil.LogQuery.RegistryInputFormat | COMRegist |

| TEXTLINE | MSUtil.LogQuery.TextLineInputFormat | COMTextLi |
|----------|---------------------------------------|-----------|
| TEXTWORD | MSUtil.LogQuery.TextWordInputFormat | COMTextW |
| TSV | MSUtil.LogQuery.TSVInputFormat | COMTSVIn |
| URLSCAN | MSUtil.LogQuery.URLScanLogInputFormat | COMURLS |
| W3C | MSUtil.LogQuery.W3CInputFormat | COMW3CI1 |
| XML | MSUtil.LogQuery.XMLInputFormat | COMXMLI |

After instantiating an input format object, an application can set the input format parameters and use the object as an argument to the <u>Execute</u> or <u>ExecuteBatch</u> methods of the <u>LogQuery</u> object.

Methods

The Input Format objects do not expose methods.

Properties

The Input Format objects expose read/write properties with the same names and capitalization as the parameters accepted by the corresponding Log Parser input format.

For example, the **MSUtil.LogQuery.EventLogInputFormat** input format object exposes a "resolveSIDs" property that controls the <u>resolveSIDs</u> parameter of the EVT input format.

The value type accepted and returned by an input format object property depends on the nature of the values that can be specified for the input format parameter, as described by the following table:

| | Property
value | |
|------------------|-------------------|-----------------|
| Parameter values | type | JScript Example |

| "ON"/"OFF" values | Boolean | oEVTInputFormat.resolveSIDs
= true; |
|--|---------|--|
| Enumeration values (e.g.
"ASC"/"PRINT"/"HEX") | String | oEVTInputFormat.binaryFormat
= "PRINT"; |
| String values | String | oEVTInputFormat.stringsSep =
", "; |
| Numeric values | Number | oIISW3CInputFormat.recurse = 10; |

For more information on Input Format Parameters, see the <u>Input Formats</u> <u>Reference</u>.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

| VBScriptEXample: Format object
var oEVTInputFormat = new ActiveXObject("MSUtil.LogQuery.EventL | |
|--|--|
| palapettegemet"); | |
| Dim oEVTInputFormat | |
| 均Satsingutefo rmat parameters | |
| DEN TROUBFOSMAT.resolveSIDs = true; | |
| See also InputFormat.binaryFormat = "PRINT"; | |
| SEVEL STREET STEARES SPECT("M'SUtil.LogQuery") | |
| <u>Output Format Objects</u> | |
| I og Grater EV Type to the top the top to the top top to the top | |
| <u>C# Example</u> "SELECT * FROM System"; | |
| · · · · · · · · · · · · · · · · · · · | |

//Sexecpue@packapdiaeoeio/fs & Dog@rectiondSet// rights reserved.
vErVoRecondSetmatoleogQueSityDExecTute(strQuery, oEVTInputFormat);
oEVTInputFormat.binaryFormat = "PRINT"
oEVTInputFormat.stringsSep = ", "
oEVTInputFormat.iCheckpoint = "MyCheckpoint.lpc"

' Create query text
strQuery = "SELECT * FROM System"

' Execute query and receive a LogRecordSet Set oRecordSet = oLogQuery.Execute (strQuery, oEVTInputFormat)

Output Format Objects

Output Format objects provide programmatic access to the <u>output</u> <u>formats</u> supported by Log Parser.

Output Format objects are instantiated with the ProgId and the <u>.NET</u> <u>COM wrapper</u> class names specified in the following table:

| Output | | |
|---------------|--------------------------------------|--------------|
| Format | ProgId | .NET COM Wr |
| CHART | MSUtil.LogQuery.ChartOutputFormat | COMChartOutp |
| CSV | MSUtil.LogQuery.CSVOutputFormat | COMCSVOutpu |
| DATAGRID | MSUtil.LogQuery.DataGridOutputFormat | COMDataGridC |
| IIS | MSUtil.LogQuery.IISOutputFormat | COMIISOutput |
| NAT | MSUtil.LogQuery.NativeOutputFormat | COMNativeOut |
| SQL | MSUtil.LogQuery.SQLOutputFormat | COMSQLOutpu |
| <u>SYSLOG</u> | MSUtil.LogQuery.SYSLOGOutputFormat | COMSYSLOG(|
| TPL | MSUtil.LogQuery.TemplateOutputFormat | COMTemplateC |
| TSV | MSUtil.LogQuery.TSVOutputFormat | COMTSVOutpu |
| W3C | MSUtil.LogQuery.W3COutputFormat | COMW3COutp |
| XML | MSUtil.LogQuery.XMLOutputFormat | COMXMLOutp |

After instantiating an output format object, an application can set the output format parameters and use the object as an argument to the

ExecuteBatch method of the LogQuery object.

Methods

The Output Format objects do not expose methods.

Properties

The Output Format objects expose read/write properties with the same names and capitalization as the parameters accepted by the corresponding Log Parser output format.

For example, the **MSUtil.LogQuery.CSVOutputFormat** output format object exposes a "headers" property that controls the <u>headers</u> parameter of the CSV output format.

The value type accepted and returned by an output format object property depends on the nature of the values that can be specified for the output format parameter, as described by the following table:

| Parameter values | Property
value
type | JScript Example |
|--|---------------------------|---|
| "ON"/"OFF" values | Boolean | oCSVOutputFormat.tabs = true; |
| Enumeration values (e.g.
"ON"/"OFF"/"AUTO") | String | oCSVOutputFormat.oDQuotes
= "OFF"; |
| String values | String | oCSVOutputFormat.oTsFormat
= "yyyy-MM-dd"; |
| Numeric values | Number | oCSVOutputFormat.oCodepage
= -1; |

For more information on Output Format Parameters, see the <u>Output</u> <u>Formats Reference</u>.

Examples

JScript example:

var oLogQuery = new ActiveXObject("MSUtil.LogQuery");

VBScriptEXample:Format object

var oEVTInputFormat = new ActiveXObject("MSUtil.LogQuery.EventL

palapett 5gquet");

Dim oEVTInputFormat 均偏reates (公教) 的复数的 and a solution of the sector of the se phpSteVuepputFormat = new ActiveXObject("MSUtil.LogQuery.CSVO SeenalsearchithSet

LogOuery Object

Inpuct-omparted in the test of test of

Set SEVER public mata Ts FORMETOD' & MSMtdd' og Query. EventLogInput

PGSYApytputFormat.oCodepage = -1;

© 2004 Microsoft Corporation. All rights reserved.

//GEALECENT DIRPLUT Format object SER STOS VOLUTO ULEFERENTE TORRECTORIZED AND THE REAL POLICY OF THE RE OMaSystem";

//SEX OCHERICIPOFENAL parameters ecsecontresenter arguery, oEVTInputFormat, oCSVOutputFor **#@\$V**OutputFormat.oDQuotes = "OFF" oCSVOutputFormat.oTsFormat = "yyyy-MM-dd" oCSVOutputFormat.oCodepage = -1

' Create query text strQuery = "SELECT TimeGenerated, Message INTO Output.csv FROM System"

'Execute query

oLogQuery.ExecuteBatch strQuery, oEVTInputFormat, oCSVOutputFor mat

COM Input Format Plugins

COM Input Format Plugins are user-developed input formats that can be used with Log Parser to provide custom parsing capabilities.

Custom input formats are developed as COM objects implementing the methods of the <u>ILogParserInputContext</u> COM interface.

Once developed and registered with the COM infrastructure, custom input formats can be used with either the Log Parser scriptable COM components through the <u>Execute</u> and <u>ExecuteBatch</u> methods of the LogQuery object, or with the Log Parser command-line executable through the <u>COM</u> input format.

- <u>ILogParserInputContext Interface</u>: describes the methods that must be implemented by custom input format COM objects.
- <u>Run Time Interaction</u>: describes how Log Parser interacts with custom input format COM objects at run time.

See also:

Custom Plugins COM Input Format

© 2004 Microsoft Corporation. All rights reserved.

ILogParserInputContext Interface

Custom input formats are developed as COM objects implementing the methods of the ILogParserInputContext COM interface. A custom input format implements the methods of this interface by implementing the ILogParserInputContext interface directly, or by implementing the IDispatch (Automation) interface exposing the methods of the ILogParserInputContext interface.

Interface

| // | |
|--|--|
| // Interface GUID | |
| // | |
| | |
| Μeቲ/ንዉታ§ 867-48AB-433c-9AFD-9D78D8B1CFC7 */ | |
| DEFINE GUID(IID II.ogParserInputContext | |

| | _ 、 | ABORESSES (NOAS DECIFIED NOT ON DETAILS ON DECISION OF THE DEC |
|---------|-------------------------------|--|
| // | GetFieldCount | Returns the number of input record fields. |
| // | I <u>coa</u> ParsenInnutCo | ntextunterface inaplemented by Utoge Parser i Ingut plugins an |
| // | <u>GetFieldType</u> | Returns the type of an input record field. |
| c]
{ | a <u>Bealt AgPars</u> erInpu | itReatesthemeixt Inplanecord. |
| | pittilialue
enum FieldType | Returns the value of a field in the current input record. |
| | CloseInputInteger=1,Real=2, | Releases all the resources and performs any necessary cleanup. |

String =3,

```
Timestamp =4,
Null
        =5
```

Properties

| virtual HRESULT ST
Custom Properties
OpenInput(IN BSTR | DMETHODCALLTYPE
Custom input formats developed as
bszFromEntity) =0;
IDispatch COM objects can support custom |
|---|--|
| | DWOPD * prevention of the second seco |
| | DM&QPFDr*natFjelfanheters. |

virtual HRESULT STDMETHODCALLTYPE GetFieldName(IN DWORD fIndex, OUT BSTR *pbszFieldName) = 0;

See also: virtual HRESULT STDMETHODCALLTYPE Ructine hterection WORD findex, <u>Custom Pluging</u> DWORD *pnFieldType) = 0;

virtual HRESOUT ATCOMENTATION AND A STRUCTURE AND A STRUCTURE

virtual HRESULT STDMETHODCALLTYPE GetValue(IN DWORD fIndex, OUT VARIANT *pvarValue) =0;

virtual HRESULT STDMETHODCALLTYPE CloseInput(IN VARIANT_BOOL bAbort) =0; };
CloseInput Method

Releases all the resources and performs any necessary cleanup.

C++ Syntax

HRESULT STDMETHODCALLTYPE CloseInput(IN VARIANT_BOOL bAb ort); Script Syntax

CloseInput(bAbort);

Parameters

bAbort

A Boolean value set to *TRUE* if the query execution has been aborted, or *FALSE* if the query execution has completed successfully.

Return Value

None.

Remarks

• This is the last method invoked by Log Parser before releasing the custom input format COM object.

Examples

C++ example:

HRESULT CProcessesInputContext::CloseInput(IN VARIANT_BOOL b Abort)

VBScript example:

// Close the snapshot handle

FuiffettonhSporeshotuleBassALID_HANDLE_VALUE)

m_Glose Handle(m_h Snapshot);

```
m_hSnapshot = INVALID_HANDLE_VALUE;
```

See Alspinction

{

ILogParserInputContext Interface OpenInput Method Run Time Interaction Custom Plugins

© 2004 Microsoft Corporation. All rights reserved.

GetFieldCount Method

Returns the number of fields in the input records.

C++ Syntax

HRESULT STDMETHODCALLTYPE GetFieldCount(OUT DWORD *pnFiel ds); Script Syntax

nFields = GetFieldCount();

Return Value

An integer value containing the number of fields in the input records.

Examples

C++ example:

```
HRESULT CProcessesInputContext::GetFieldCount( OUT DWORD *pn
Fields )
```

VBScript example:

// This Input Context exports 4 fields

Function GetFieldCount()

*pnFields = 4;

' This Input Format returns 4 or 6 fields If the backdedFields = True Then

```
See also: GetFieldCount = 6
```

```
<u>ILogParSennputContext Interface</u>
<u>Run Time Interface</u>
<u>Custom Foldofns</u>
```

End Function © 2004 Microsoft Corporation. All rights reserved.

GetFieldName Method

Returns the name of an input record field.

C++ Syntax

HRESULT STDMETHODCALLTYPE GetFieldName(IN DWORD fIndex, O UT BSTR *pbszFieldName); Script Syntax

fieldName = GetFieldName(fIndex);

Parameters

fIndex

The 0-based index of the input record field. The index value is guaranteed to be smaller than the number of fields returned by the <u>GetFieldCount</u> method.

Return Value

A string value containing the name of the input record field at the specified position.

Examples

C++ example:

HRESULT CProcessesInputContext::GetFieldName(IN DWORD fIndex, OUT BSTR *pbszFieldName) V**BScript example:**

```
Function GetFieldName(nFieldIndex)
          Select Case nFieldIndex
               Case "pbszFieldName = SysAllocString(L"ImageName");
                     ereakieldName = "QFE"
See also:
<u>ILogParserInputContext Interface</u>
CetFieldType MetroFieldName = "Description"
Run Time Int
                     GetFieldName = SysAllocString(L"PID");
Custom Plugins
                Case Break;
                  GetFieldName = "InstalledBy"
<u>2004 Microsoft Corporation. All rights reserved.</u>
               2: {GetFieldName = "Comments"
Case $PbszFieldName = SysAllocString(L"ParentPID");
          case 2:
                     break:
GetHieldName = "SP"
         End Select
    End Function {
                     *pbszFieldName = SysAllocString(L"Threads");
                     break;
                  }
       }
       return S_OK;
    }
```

GetFieldType Method

Returns the type of an input record field.

C++ Syntax

HRESULT STDMETHODCALLTYPE GetFieldType(IN DWORD fIndex, OU T DWORD *pnFieldType); Script Syntax

fieldType = GetFieldType(fIndex);

Parameters

fIndex

The 0-based index of the input record field. The index value is guaranteed to be smaller than the number of fields returned by the <u>GetFieldCount</u> method.

Return Value

An integer value from the *FieldType* enumeration containing the Log Parser <u>data type</u> of the input record field at the specified position.

Examples

C++ example:

HRESULT CProcessesInputContext::GetFieldType(IN DWORD fIndex, OUT DWORD *pnFieldType) V**BScript example:**

```
Function GetFieldType(nFieldIndex)
         Select Case nFieldIndex
              Case<sup>/</sup><sup>0</sup> ImageName</sup>
                   *gnFieldType = ILogParserInputContext::String;
See also:
                    ontextdTyphace
ILogParserInput
GetFieldName<sup>1</sup>Method
Run Time Interactioning
                   GetFieldType = 3
Custom Plugins
              Case<sup>//2</sup>P
                   * pnFieldType = ILogParserInputContext::Integer;
              © 200 200 Corporation. All rights reserved.
              Case 3
                   ' String
         case 2:
                   GetFieldType = 3
              Case<sup>//</sup>4<sup>ParentPID</sup>
                   *gnFieldType = ILogParserInputContext::Integer;
                   GetFieldType = 3
              Case 5
                   ' String
         case 3:
                   GetFieldType = 3
                   // Threads
         End Select*pnFieldType = ILogParserInputContext::Integer;
                   break;
   End Function
      }
      return S_OK;
    }
```

GetValue Method

Returns the value of an input record field.

C++ Syntax

HRESULT STDMETHODCALLTYPE GetValue(IN DWORD fIndex, OUT V ARIANT *pvarValue); Script Syntax

value = GetValue(fIndex);

Parameters

fIndex

The 0-based index of the input record field. The index value is guaranteed to be smaller than the number of fields returned by the <u>GetFieldCount</u> method.

Return Value

A VARIANT containing the value of the specified field.

The VARIANT type must match the Log Parser <u>data type</u> declared by the <u>GetFieldType</u> method, as shown in the following table:

| Declared
Field Type | C++ VARIANT Type | VBScript
Type |
|------------------------|--------------------------------|------------------|
| INTEGER | VT_I8 (also compatible: VT_I4) | Long
(VT_I4) |
| REAL | VT_R8 | Double |

| | | (VT_R8) |
|-----------|--|---------------------|
| STRING | VT_BSTR | String
(VT_BSTR) |
| TIMESTAMP | VT_DATE (also compatible:
VT_I8, VT_I4 containing the
number of 100-nanosecond
intervals since January 1, year 0) | Date
(VT_DATE) |
| NULL | VT_NULL (also compatible:
VT_EMPTY) | Null
(VT_NULL) |

Remarks

- Any value can be returned as a VT_NULL or VT_EMPTY VARIANT (a Null VBScript variable) to indicate a <u>NULL</u> value, regardless of the field type declared by the <u>GetFieldType</u> method.
- Due to query execution optimizations, there is no guarantee that the GetValue method will be called for all the fields of an input record. In fact, the GetValue method will only be called for those fields that are referred to by the currently executing query.

For example, if a query refers to two fields only out of an input record made up of ten fields, then the GetValue method will be called for those two fields only.

If a query does not refer to any input record field (e.g. "SELECT COUNT(*)"), then the GetValue method will never be called.

Examples

C++ example:

HRESULT CProcessesInputContext::GetValue(IN DWORD fIndex, OUT VARIANT *pvarValue)

VBScript example:

// Initialize return value

Fulletiont the (Appled Index)

Select Case nFieldIndex switch(fIndex)

See also ase 0

```
<u>ILogParsenteputContext Interface</u>
<u>ReadRecord Method</u>
(magninet) - WT_DCTD:
Run Time<sup>a</sup> Anteraction (pvarValue) = VT_BSTR;
Custom Plugins ription R( pvarValue ) = SysAllocString( m_processEntry32.
   szExeF@etValue = m_objQFEArray(m_nIndex).Description
        Case 2 break;
           ' Instandar Microsoft Corporation. All rights reserved.
           GetValue = m objQFEArray(m nIndex).InstallDate
        case k {
           'Installe
           GetValue Inprobleme Array mI4, Index). Installed By
        Case 4 V_I4( pvarValue ) = m_processEntry32.th32ProcessID;
           ' Condifications
           GetValue = m objQFEArray(m nIndex).FixComments
        Case 5
        cases 2: {
           GetVallerentP_bjQFEArray(m_nIndex).ServicePackInEffect
               V VT( pvarValue ) = VT I4;
      End Select/_I4( pvarValue ) = m_processEntry32.th32ParentProcessID
   End Function Freak;
             }
        case 3: {
               // Threads
                V VT( pvarValue ) = VT I4;
                V I4( pvarValue ) = m processEntry32.cntThreads;
                break;
```

```
}
}
return S_OK;
}
```

OpenInput Method

Processes the specified <u>from-entity</u> and performs any necessary initialization.

C++ Syntax

HRESULT STDMETHODCALLTYPE OpenInput(IN BSTR bszFromEntity);

Script Syntax

OpenInput(bszFromEntity);

Parameters

bszFromEntity

The from-entity specified in the <u>FROM</u> clause of the currently executing query, or an empty string if Log Parser is executed in <u>Help</u> <u>Mode</u> to display the quick-reference help on the custom input format.

Return Value

None.

Remarks

• The OpenInput method is the first method called by Log Parser after the custom input format COM object has been instantiated. An implementation of this method would usually perform any necessary object initialization, prepare the from-entity for input record retrieval (e.g. opening an input file), and eventually pre-process the input to gather the input record fields meta-information that will be returned by the <u>GetFieldCount</u>, <u>GetFieldName</u>, and <u>GetFieldType</u> methods.

 Users can execute the Log Parser command-line executable in <u>Help</u> <u>Mode</u> to display a quick-reference help on a custom input format. The quick-reference help displays the input record field names and types, which are retrieved through calls to the <u>GetFieldCount</u>, <u>GetFieldName</u>, and <u>GetFieldType</u> methods.

If the user-supplied help mode command does not include a fromentity, the *bszFromEntity* argument wil be an empty string. In these cases, a custom input format COM object can behave in two ways:

- If the input record fields do not depend on the from-entity specified in the query (i.e. if the input record structure is fixed), then the custom input format COM object should accept the empty from-entity without returning an error, allowing Log Parser to subsequently call the <u>GetFieldCount</u>, <u>GetFieldName</u>, and <u>GetFieldType</u> methods to retrieve the input record structure;
- If the input record fields depend on the from-entity specified in the query (i.e. if the input record structure is extracted from the input data), then the custom input format COM object should reject the empty from-entity returning an error, which will in turn cause the help command to display a warning message to the user in place of the input record structure.

Examples

C++ example:

HRESULT CProcessesInputContext::OpenInput(IN BSTR bszFromEntit y)

VBScript example:

// Initialize object

Function OpenInput(strComputerName)

See also im nLength

ILogPareenImpSitOKntext Interface

<u>CloseInpDefaethcomputer</u> name is local machine <u>Run Time IsNeda(sticio</u>mputerName) Or Len(strComputerName) = 0 Then <u>Custom Plugitrs</u> omputerName = "."

End If

© 2004 Microsoft Corporation. All rights reserved.

' Query for all the QFE's on the specified machine Set objWMIService = GetObject("winmgmts:" & "{impersonationL evel=impersonate}!\\" & strComputerName & "\root\cimv2") Set obiOEEs = obiWMIService ExecQuery ("Select * from Win32)

```
Set objQFEs = objWMIService.ExecQuery ("Select * from Win32_
QuickFixEngineering")
```

' Store in array m_objQFEArray = Array() For Each objQFE In objQFEs ReDim Preserve m_objQFEArray(UBound(m_objQFEArray)

+1)

```
Set m_objQFEArray( UBound(m_objQFEArray) ) = objQFE
Next
```

m_nIndex = LBound(m_objQFEArray)

End Function

ReadRecord Method

Reads the next input record.

C++ Syntax

HRESULT STDMETHODCALLTYPE ReadRecord(OUT VARIANT_BOOL *pbDataAvailable); Script Syntax

bDataAvailable = ReadRecord();

Return Value

A Boolean value set to *TRUE* if a new input record has been read and is available for consumption, or *FALSE* if there are no more input records to return.

Remarks

- An implementation of the ReadRecord method would usually read a new data item from the input and store it internally, waiting for Log Parser to subsequently call the <u>GetValue</u> method multiple times to retrieve the input record field values.
- The Boolean value returned by the ReadRecord method is used by Log Parser to determine which custom input format methods will be called next.

If the method returns *TRUE*, signaling availability of an input record, Log Parser will call the <u>GetValue</u> method multiple times to retrieve the input record field values, followed by a new call to the ReadRecord method to read the next input record.

If the method returns *FALSE*, signaling the end of the input data, Log Parser will call the <u>CloseInput</u> method and release the custom input format COM object.

Examples

C++ example:

```
HRESULT CProcessesInputContext::ReadRecord(OUT VARIANT BOO
   L *pbDataAvailable)
  VBScript example:
     if( m hSnapshot == INVALID HANDLE VALUE )
   Function ReadRecord()
       // This is the first time we have been called
     If m nIndex >= UBound(m objQFEArray) Then
        /Ænghaeshapshærafithæaurrent processes
See alsgehmenshot=FareateToolhelp32Snapshot(TH32CS_SNAPPROC
ESSLO);
ILogParserInputContext Interface
INdmanSmapshot == INVALID_HANDLE_VALUE)
GetValue Wethod
Run Time Interaction_nIndex + 1
Custom Products of True
     End Jeturn HRESULT_FROM_WIN32( GetLastError() );
   , <u>© 2004 Microsoft Corporation. All rights reserved</u>.
End Function
       // Get the first entry
       if( !Process32First( m_hSnapshot, &m;_processEntry32 ) )
        ł
          DWORD dwLastError = GetLastError();
          if( dwLastError == ERROR_NO_MORE_FILES )
          {
            // No processes
            *pbDataAvailable = VARIANT FALSE;
            return S OK;
          }
          else
          {
            // Error
            return HRESULT FROM WIN32(GetLastError());
          }
```

```
}
else
{
   // There is data available
   *pbDataAvailable = VARIANT_TRUE;
   return S_OK;
  }
}
else
{
```

 $\ensuremath{/\!/}$ We have already been called before, and we have already taken a s napshot

```
// Get the next entry
  if( !Process32Next( m_hSnapshot, &m;_processEntry32 ) )
  {
    DWORD dwLastError = GetLastError();
    if( dwLastError == ERROR_NO_MORE_FILES )
    {
      // No more processes
      *pbDataAvailable = VARIANT_FALSE;
      return S_OK;
    }
    else
    {
      // Error
      return HRESULT_FROM_WIN32( GetLastError() );
    }
  }
  else
  {
    // There is data available
    *pbDataAvailable = VARIANT_TRUE;
    return S_OK;
  }
}
```

}

Custom Properties

Provide parameters for the custom input format.

C++ Syntax

HRESULT STDMETHODCALLTYPE put_propertyName(IN VARIANT *val ue); Script Syntax

put_propertyName(value);

Parameters

value

A VT_BSTR VARIANT containing the string parameter value specified with the <u>-iCOMParams</u> parameter of the <u>COM</u> input format.

Return Value

None.

Remarks

- Custom properties can only be exposed by custom input formats that implement the **IDispatch** (Automation) interface. These are usually custom input formats developed as *scriptlets* (.wsc files) written in JScript or VBScript.
- Custom properties exposed by a custom input format can be set in two different ways:
 - With the Log Parser command-line executable, custom properties can be set through the <u>-iCOMParams</u> parameter of the <u>COM</u> input

format, as shown in the following example:

```
C:\>LogParser "SELECT * FROM file.txt" -i:COM -iProgID:MySample.C
-iCOMParams:property1=value1,property2=value2
```

 With the Log Parser scriptable COM components, custom properties can be set directly on the custom input format object before specifying the object as an argument to the <u>Execute</u> or <u>ExecuteBatch</u> methods of the <u>LogQuery</u> object, as shown in the following JScript example:

var objLogQuery = new ActiveXObject("MSUtil.LogQuery");

// Create custom input format object

var objCustomInputFormat = new ActiveXObject("MySample.CustomIn
ExampleSormat");

VB\$Gript.example.it format parameters objCustomInputFormat.property1 = "value1"; FuobjConsport_lexpetFiledFilelpls(perVgRue)"value2";

© 2004 Microsoft Corporation. All rights reserved.

Run Time Interaction

Custom input format COM objects are used by Log Parser in two different scenarios: when executing a query, and when displaying a quick-reference help on the custom input format when the Log Parser command-line executable is used in <u>Help Mode</u>.

Query Execution Scenario

In this scenario, a custom input format COM object is used to retrieve input records from the specified <u>from-entity</u>.

To make an example of the sequence of the method calls invoked by Log Parser on the custom input format COM object in this scenario, we will assume that the custom input format generates input records containing the following four fields:

- "FirstField", STRING type;
- "SecondField", INTEGER type;
- "ThirdField", TIMESTAMP type;
- "FourthField", STRING type.

In addition, we will assume that the query being executed references only three fields out of the four fields exported by the custom input format, as in the following example:

SELECT FourthField, ThirdField

FROM InputFile.txt

WHERE FirstField LIKE '%test%'

The following table shows the sequence of method calls under these assumptions:

| Method call | Returned value | Returned value
description |
|----------------------------|----------------|-------------------------------|
| Object is instantiated | | |
| OpenInput("InputFile.txt") | None | |
| GetFieldCount() | 4 | |
| GetFieldName(0) | "FirstField" | |

| GetFieldType(0) | 3 | FieldType.String |
|-----------------|--------------------|------------------------------|
| GetFieldName(1) | "SecondField" | |
| GetFieldType(1) | 1 | FieldType.Integer |
| GetFieldName(2) | "ThirdField" | |
| GetFieldType(2) | 4 | FieldType.Timestamp |
| GetFieldName(3) | "FourthField" | |
| GetFieldType(3) | 3 | FieldType.String |
| ReadRecord() | TRUE | an input record is available |
| GetValue(0) | VT_BSTR
VARIANT | first field value |
| GetValue(2) | VT_DATE
VARIANT | third field value |
| GetValue(3) | VT_BSTR
VARIANT | fourth field value |
| ReadRecord() | TRUE | an input record is available |
| GetValue(0) | VT_BSTR
VARIANT | first field value |
| GetValue(2) | VT_DATE
VARIANT | third field value |

| GetValue(3) | VT_BSTR
VARIANT | fourth field value |
|--------------------|--------------------|------------------------------------|
| | | |
| | | |
| ReadRecord() | TRUE | an input record is
available |
| GetValue(0) | VT_BSTR
VARIANT | first field value |
| GetValue(2) | VT_DATE
VARIANT | third field value |
| GetValue(3) | VT_BSTR
VARIANT | fourth field value |
| ReadRecord() | FALSE | no more input records
available |
| CloseInput(FALSE) | None | |
| Object is released | | |

Help Mode Scenario

When the Log Parser command-line executable is used in <u>Help Mode</u> to display a quick-reference help on the custom input format, the custom input format COM object is only used to retrieve the field information that is displayed to the user.

The user-supplied help mode command may or may be not include a from-entity, as shown in the following examples:

C:\>LogParser -h -i:COM -iProgID:MySample.CustomInputFormat file.txt

C:\>LogParser -h -i:COM -iProgID:MySample.CustomInputFormat

If the user-supplied help mode command does not include a from-entity, then the *bszFromEntity* argument of the <u>OpenInput</u> method will be an empty string. See the Remarks section of the <u>OpenInput Method</u> <u>Reference</u> for more information on how custom input format COM objects should behave in this case.

To make an example of the sequence of the method calls invoked by Log Parser on the custom input format COM object in this scenario, we will assume that the custom input format generates input records containing the following four fields:

- "FirstField", STRING type;
- "SecondField", INTEGER type;
- "ThirdField", TIMESTAMP type;
- "FourthField", STRING type.

In addition, we will assume that the help command does not include a from-entity.

The following table shows the sequence of method calls under these assumptions:

| Method call | Returned value | Returned value description |
|------------------------|----------------|----------------------------|
| Object is instantiated | | |
| OpenInput("") | None | |
| GetFieldCount() | 4 | |
| GetFieldName(0) | "FirstField" | |
| GetFieldType(0) | 3 | FieldType.String |
| GetFieldName(1) | "SecondField" | |
| GetFieldType(1) | 1 | FieldType.Integer |
| GetFieldName(2) | "ThirdField" | |
| GetFieldType(2) | 4 | FieldType.Timestamp |
| GetFieldName(3) | "FourthField" | |
| GetFieldType(3) | 3 | FieldType.String |
| CloseInput(FALSE) | None | |
| Object is released | | |

See also:

ILogParserInputContext Interface Custom Plugins

© 2004 Microsoft Corporation. All rights reserved.

Legal Information

Microsoft Documentation

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2004 Microsoft Corporation. All rights reserved.

Active Directory, JScript, Microsoft, MSDN, Visual Basic, Visual Studio, Windows, Windows Media, and Windows Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.