User-Defined Logs

This document explores how to create user-defined logs for use with **MySQL::Log::ParseFilter** and **mysqlsla v2**. A udl defines its own record separator, headers and values. Logs created with **MySQL Proxy** are good examples of user-defined logs. The MySQL::Log::ParseFilter (MLP) module and mysqlsla v2 which implements it support parsing and filtering user-defined logs.

This document is written using mysqlsla as the example but the information applies equally to any script that uses the MySQL::Log::ParseFilter (MLP) module.

Defining a udl is not a trivial task and requires a good knowledge of **Perl regular expressions**. Regex is used to parse and extract **meta-properties** and their values from the log.

User-Defined Logs Table of Contents

For mysqlsla v2, defining the udl is only the first step: afterwards, you will need to create a **custom report** to actually present the log values in the way that suites your needs. (This does not apply to MLP in general; it is only a mysqlsla v2 feature).

UDL Format File

« Top

A udl is defined by a format given in a file: the udl format file. This format file itself has a specific template imposed by the MLP module. With mysqlsla, the udl format file is given with the **udl-format** option.

The udl format file template is:

```
record separator
header line format
header line values
```

The record separtor (rs for short) must be the first line and only one, single line. The record separtor tells MLP how/where to seperator queries in the log. (This is the same **Perl input record separator**.)

Although the rs in the udl format file must be a single line, it can still create a multi-line rs when used by mysqlsla because a literal \n in the rs is made into a real newline. Tabs can also be used because \t in the rs is made into a real tab.

The rs should be well-defined, distinct and easy for MLP to split the queries in the log on it. This is discussed later in the section **A Well-Defined UDL**.

After the first line (the rs) all other lines are header format/values line pairs. Almost every MySQL log prefixes queries with a header that provide meta-property values. The slow query log is the classic example. (The general log is the counter-example; its format does not fit a udl.) User-defined logs use the same construct: a series of header lines preceding each block of SQL statements.

Therefore, only header format/values line pairs must be defined in the udl format file. MLP will treat everything after these header lines as SQL statements until the start of the next series of header lines.

The first line in the pair, the header format, is a Perl regex expression without m//. It must be contained on a single line; it cannot be escaped and written on multiple lines. This regex expression is not changed in anyway. If for example the pattern 'db=(.*)' is given, then MLP matches the appropriate header line from the log like: $line =~ /db=(.*)/. Note that the patterns are case sensitive.

Note also that capture buffers must be used. The in example pattern above there is only one: (.*). Since you know Perl regex, you know that this will cause Perl to capture the value of whatever matched at that point and save it in an appropriately numbered match variable ($1 in this case).

The capture buffers in the header format line must match in number and order the header values given in the header values line.

The second line in the pair, the header values, is the single-space-separated list of typed meta-property names in the form: name:type. (**type** is described in the next section). You can name them anything you want (limit their names to alpha-numeric and _ to be safe). These meta-properties must correspond in number and order to the caputre buffers in the header format line. Therefore, if the header line used a pattern with three capture buffers, for example 'cid=(\d+) time=([\d\.]+) db=(\w+)', then three meta-properties must be give, for example 'cid:n t:na db:s'.

If no header format/values line pairs are given, MLP will treat everything in the log as SQL statements, splitting them with the given record separator. On the other hand, you can define as many headers as you want. Currently there is no way to define blank headers or conditional headers. If you need such a construct in your user-defined log, use zero or special values and filter out those queries later with appropriate meta-filter conditions.

Meta-Property Types

« Top

Every given meta-property name must be typed. In order to allow proper filtering on a meta-property, MLP must know it basic type. Currently, there are five types.

n (Numeric)

« Top

n type meta-properties are numeric and only the first occurrence of their value for each unique query is saved. They are most frequently used for filtering *during* log parsing (or **replay** re-filtering).

An example of this type of meta-property is a query's connection ID (`cid` in general, binary and msl logs). Although a single unique query may have many different connection IDs, it is generally only used

to filter queries during parsing in order to isolate the activity of a specific connection ID (meta-filter="cid=123" for example).

Nothing more about n type meta-properties is saved or calculated.

na (Numeric with Aggregation)

« Top

na type meta-properties are the most common type and probably what you are looking for. They are numeric with "aggregation" which means that MLP automatically creates and saves four additional meta-properties: name_min, name_max, name_avg, name_sum (where 'name' is the given meta-property name).

It is important to note first that for na type meta-properties, the base meta-property name itself (just 'name') is only available during log parsing as a meta-filter; it is not ultimately saved with the query and its other meta-properties. This is like exactly like slow log meta-properties `t` and `l`. Then, *after* log parsing, the aggregate names (name_min, name_max, name_avg, name_sum) are calculated, saved, and then filtered accordingly (and can also be used with **sort**).

nf (Numeric with Full Aggregation)

« Top

nf type meta-properties are na type meta-properties with one extra meta-property created and saved by MLP: name_all.

name_all will be an array of every single value for name for each unique query in the order that they occurred. These "all values" (or "full aggregate values") are required if you want to calculate **nth-percent** or **dist** with mysqlsla.

At present, MLP saves these values in an array, even redundant values. In a future version, these values may be saved in a hash with one meta-value per key and each key's value being the count of that meta-value. That way, 100 zero values in an array are reduced to a single hash key-value pair (meta-value => 100).

s (String)

« Top

s type meta-properties are the string equivalents of n type meta-properties. Only the first occrence of their value for each unique query is saved and they are used for filtering during log parsing. Nothing more about them is created or saved by MLP.

u (User)

« Top

u type meta-properties are a special type of meta-property. They are meant to represent "user@host IP" values. In truth, they can represent any value (the database used by the query, for example) if you understand their function inside MLP.

Inside MLP, u type meta-properites are saved in a hash called user. Each unique user for each unique query is saved in this hash which tracks how many times each user executed the given query. Furthemore, these per-query users are also added to the global users hash which tracks user-query usage globally (log-wide).

In a **mysqlsla v2 report**, this type of meta-property is later recalled and printed by using the special summary value called users.

In truth, MLP does not care what value you actually use for a u type meta-property. It will store its unique occurrances and per-query and global counts all the same. But, the general purpose is to use a u type meta-property for something like the query's "user@host IP".

A Well-Defined UDL

« Top

A well-defined udl is essential. If you have control over the definition of your user-defined log, take the time to define it well so that it is quick and easy for MLP to parse.

MLP reads the user-defined log in chunks called records. Records are divided by the record separator. Each record will begin with its header if any were defined, followed by what should be SQL statements.

The first step then in a well-defined udl is the record separator. An rs of ';\n' will not work unless you are 100% sure that your udl will have only single-statement queries: headers followed by only one semicolon-terminated SQL statment.

Otherewise, with mutli-statement log entries, like what is seen often in slow and binary logs, you should create a simple and distinct record separator. For example: '\n#\n'. This rs would easily divide a log like:

```
# header 1
# header 2
SQL 1
SQL 2
#
# header 3
# header 4
SQL 3
```

For such a log, MLP would read two records: the first containing header 1 and 2 and SQL 1 and 2, and the second containg header 3 and 4 and SQL 3 (although the headers are numbered differently here, they would in fact be the same differing only in their meta-values for the SQL statements that follow them).

The second step in a well-defined udl is the headers themselves. # is commonly used to mark a header line. Because records include both headers and SQL statements, MLP must further divide the one from the other. Therefore, to make this easy, define easily identifiable header lines such as those that begin with # because no SQL statement should ever begin with #.

Third and finally, a well-defined udl uses simply defined headers. Remember: the log is meant to be read by a

computer, not a human. The reports that mysqlsla can make are what we humans read. Plus, you have to write the regex patterns to match the headers, so make them simple for you and the computer by being consistent with punctuations, spaces, tabs, and captialization (because everything is case-sensitive). Also, in my opinion, more is less in the udl: be terse but well documented and leave wordiness to the report; this reduces typos (it is a lot more difficult to make a typo with 're' than 'rows_examined').

Quick Example

« Top

This udl format file:

```
\n#\n
cid:(\d+) rows:(\d+) time:([\d\.]+)
cid:n rows:na time:nf
host:(.*)
users:u
```

parses this user-defined log:

```
# cid:3 rows:33 time:1.100040
# host:foo.server.com
SELECT * FROM table WHERE col IN (1, 2, 3);
#
# cid:4 rows:44444 time:0.445000
# host:192.168.0.1
SET something;
UPDATE table SET foo = bar WHERE bananas = 1;
SELECT col FROM table GROUP BY fruit;
#
# cid:3 rows:5 time:5.500301
# host:localhost
USE database;
SELECT col FROM table WHERE col = 1;
```

making the following meta-properties available:

- cid
- rows
- rows_min
- rows_max
- rows_avg
- rows_sum
- time
- time_min
- time_max
- time_avg
- time_sum
- time_all
- users

User-Defined Logs was last updated July 12, 2008 for MySQL::Log::ParseFilter v1.00 and mysqlsla v2.00.