

# Adding Headers-Quotes-Line Feeds to Eventgen Events

James H Baxter

Jan 17, 2022

## Introduction

This code was developed to modify events which were originally ingested via the **Splunk Add-on for Linux and Unix** and have been exported from a Splunk search so that they can be used with Splunk Eventgen to create artificial events of this type.

When you export sample Linux/Unix OS-related events (cpu, disk, interfaces, etc.) from Splunk, you get events with just the data fields. What isn't obvious at first is that when these events were generated & sent to Splunk indexers by a forwarder, they originally included a header (for each event) and extra line feeds. When you see the events in a Splunk search, they only include the data fields - no header (vstat being the only exception I'm aware of) - but the fields have been properly parsed and identified (CPU & such) and appear in the left-hand 'Interesting Fields' list in Splunk (using Smart or Verbose Mode). An initial investigation of the props/transforms in the Add-on doesn't make it obvious where/how these fields were extracted or identified - but there are field aliases and evals for creating additional fields. It's only when you look at the scripts in /bin (cpu.sh, for example) that you see that each event was created with a smaller number of specific fields, and included a header. In many cases, there is also an extra line feed added. Doing a comparison of the various .sh files in /bin and another look through the props file and this all starts to make sense.

So - after exporting some sample Linux/Unix events from Splunk, you have to edit those events to add the header and extra line feeds - recreating the events in the format they were in when sent to Splunk for indexing - before you can use them with Eventgen to create artificial events. Otherwise, the props/transforms from the Splunk Add-on for Linux/Unix will not properly parse the events.

This code adds the headers and extra line feeds to events from a provided sample file, and creates a new output sample file (so that the origin file is not modified). It also adds quotes around the events, which may or may not be needed but doesn't seem to hurt.

The other problem with these Linux/Unix events is that their timestamps (as they appear in Splunk searches) were created at index-time - the events themselves did not include a timestamp. When you export these events for use with Eventgen, they again do not include a timestamp. There are two ways of dealing with this situation that I'm aware of:

1. Use the mode = sample option in eventgen.conf, which blasts all of the events in your sample file out at once; Splunk will give them an index-time timestamp - but they all have the same timestamp per generation interval. Example:

```
[linux_os_events_10000.csv]
disabled = 0
mode = sample
timeField = _time
sampletype = csv
interval = 60
earliest = -60s
```

```
latest = now
```

2. My preference is to use mode = replay and the timeField = ``\_time`` option, which requires you to include the ``\_time`` field in the Splunk export:

```
[linux_os_events_10000.csv]
disabled = 0
mode = replay
timeField = _time
sampletype = csv
interval = 60
earliest = -60s
latest = now
```

The search for exporting these events from Splunk is (substituting the correct index(es) for your environment):

```
index=linux_os
| reverse
| table index,host,source,sourcetype,_time,_raw
```

Exclude the \_time column if you are not going to use this approach. Instead, use:

```
index=linux_os
| reverse
| table index,host,source,sourcetype,_raw
```

and remove the timeField = \_time entry from the stanza.

## Python Code

```
import csv
```

```
inFile = "linux_unix_addon_events_10000.csv"
outFile = "linux_unix_addon_events_10000_formatted.csv"
```

```
# The first (commented out) df is from version 8.4.0 of the add-on; the second is from version 8.3.1
# The 'Siize' field in the v8.3.1 df header is what is in the df.sh script
```

```
# vmstat is commented out in the list below because those events already have headers in the Splunk export.
# Note that the spacing between the various header fields must be maintained exactly for parsing to work.
```

```
headers = {
    "bandwidth": "Name  rxPackets_PS txPackets_PS rxKB_PS txKB_PS",
    "cpu": "CPU  pctUser  pctNice pctSystem pctlowait  pctIdle",
    "cpu_metric": "CPU  pctUser  pctNice pctSystem pctlowait  pctIdle  OSName  OS_version",
    "IP_address": "IP_address",
    "# df": "Filesystem\tType\tSize\tUsed\tAvail\tUsePct\tInodes\tUsed\tIfree\tUsePct\tMountedOn",
    "df": "Filesystem\tType\tSiize\tUsed\tAvail\tUsePct\tMountedOn",
}
```



```

with open(inFile) as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        sourcetype = row[3]

        # Print the sample file header row as-is
        # index,host,source,sourcetype,_time,_raw
        if line_count == 0:
            lineout = ",".join(row)
            lineout = lineout + "\n"
            fo.write(lineout)
            line_count += 1
            continue

        # Strip the milliseconds & TZ portion of the timestamp off, if present
        # since Eventgen doesn't seem to process those correctly even with good regex,
        # and timestamps from this source don't appear to use milliseconds anyway
        # 2017-08-29T01:12:50.000+0000 to 2017-08-29T01:12:50

        # if using a _time column in the source file, _raw is row[5]
        if len(row) == 6:

            # in this case, row[4] is _time column
            timestamp = row[4]
            if '.' in timestamp:
                idx = timestamp.index('.')
                timestamp = timestamp[:idx]
            row[4] = timestamp
            rawdata = row[5]

        # len(row) != 6 so assuming index,host,source,sourcetype,_raw
        # and _raw is in row[4]
        else:
            rawdata = row[4]

        # process this row's _raw data
        row[5] = add_header_quotes_linefeeds(sourcetype, rawdata)
        lineout = ",".join(row)
        if DEBUG: print(lineout)
        fo.write(lineout)
        line_count += 1

    if DEBUG: print(f'Processed {line_count} lines.')

fo.close()

```

END