



Time functions

earliest(<value>)

Description

Returns the chronologically earliest seen occurrence of a value in a field.

Usage

You can use this function with the [chart](#), [mstats](#), [stats](#), [timechart](#), and [tstats](#) commands.

This function processes field values as strings.

Basic example

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to [get the tutorial data into Splunk](#). Use the time range All time when you run the search.

You run the following search to locate invalid user login attempts against a sshd (Secure Shell Daemon). You use the table command to see the values in the `_time`, `source`, and `_raw` fields.

```
sourcetype=secure invalid user "sshd[5258]" | table _time source _raw
```



The results appear on the Statistics tab and look something like this:

_time	source	_raw
2023-05-01 00:15:05	tutorialdata.zip:./mailsv/secure.log	Mon May 01 2023 00:15:05 mailsv1 sshd[5258]: Failed password for invalid user tomcat from 67.170.226.218 port 1490 ssh2
2023-04-30 00:16:17	tutorialdata.zip:./www2/secure.log	Sun Apr 30 2023 00:16:17 www2 sshd[5258]: Failed password for invalid user brian from 130.253.37.97 port 4284 ssh2
2023-04-30 00:11:25	tutorialdata.zip:./www3/secure.log	Sun Apr 30 2023 00:11:25 www3 sshd[5258]: Failed password for invalid user operator from 222.169.224.226 port 1711 ssh2
2023-04-29 00:19:01	tutorialdata.zip:./www1/secure.log	Sat Apr 29 2023 00:19:01 www1 sshd[5258]: Failed password for invalid user rightscale from 87.194.216.51 port 3361 ssh2
2023-04-29 00:13:45	tutorialdata.zip:./mailsv/secure.log	Sat Apr 29 2023 00:13:45 mailsv1 sshd[5258]: Failed password for invalid user testuser from 194.8.74.23 port 3626 ssh2
2023-04-28 00:23:28	tutorialdata.zip:./www1/secure.log	Fri Apr 28 2023 00:23:28 www1 sshd[5258]: Failed password for invalid user redmine from 91.208.184.24 port 3587 ssh2

You extend the search using the `earliest` function.

```
sourcetype=secure invalid user "sshd[5258]" | table _time source _raw | stats earliest(_raw)
```

—

The search returns the event with the `_time` value `2023-04-28 00:23:28`, which is the event with the oldest timestamp.

_time	source	_raw
2022-04-28 00:23:28	tutorialdata.zip:./www1/secure.log	Fri Apr 28 2023 00:23:28 www1 sshd[5258]: Failed password for invalid user redmine from 91.208.184.24 port 3587 ssh2

earliest_time(<value>)

Description

Returns the UNIX time of the chronologically earliest-seen occurrence of a given field value.

Usage

You can use this function with the [mstats](#), [stats](#), and [tstats](#) commands.

This function processes field values as strings.

If you have metrics data, you can use `earliest_time` function in conjunction with the `earliest`, `latest`, and `latest_time` functions to calculate the rate of increase for a counter. Alternatively you can use the `rate` counter to do the same thing.

Basic example

The following search runs against metric data. It is designed to return the earliest UNIX time values on every minute for each `metric_name` that begins with `deploy`.

```
| mstats earliest_time(_value) where index=_metrics metric_name=deploy*
BY metric_name span=1m
```

—

The results appear on the Statistics tab and look something like this:

_time	metric_name	earliest_time(_value)
2023-12-18 09:30:00	deploy-connections.nCurrent	1702920600.000000
2023-12-18 09:31:00	deploy-connections.nStarted	1702920660.000000
2023-12-18 09:32:00	deploy-server.volumeCompletedKB	1702920720.000000
2023-12-18 09:33:00	deploy-connections.nCurrent	1702920780.000000
2023-12-18 09:34:00	deploy-connections.nStarted	1702920840.000000
2023-12-18 09:35:00	deploy-server.volumeCompletedKB	1702920900.000000

latest(<value>)

Description

Returns the chronologically latest seen occurrence of a value in a field.

Usage

You can use this function with the [chart](#), [mstats](#), [stats](#), [timechart](#), and [tstats](#) commands.

This function processes field values as strings.

Basic example

This example uses the sample data from the Search Tutorial. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to [get the tutorial data into Splunk](#). Use the time range All time when you run the search.

You run the following search to locate invalid user login attempts against a specific sshd (Secure Shell Daemon). You use the table command to see the values in the `_time`, `source`, and `_raw` fields.

```
sourcetype=secure invalid user "sshd[5258]" | table _time source _raw
```

○

The results appear on the Statistics tab and look something like this:

_time	source	_raw
2023-05-01 00:15:05	tutorialdata.zip:./ mailsv/secure.log	Mon May 01 2023 00:15:05 mailsv1 sshd[5258]: Failed password for invalid user tomcat from 67.170.226.218 port 1490 ssh2
2023-04-30 00:16:17	tutorialdata.zip:./ www2/secure.log	Sun Apr 30 2023 00:16:17 www2 sshd[5258]: Failed password for invalid user brian from 130.253.37.97 port 4284 ssh2
2023-04-30 00:11:25	tutorialdata.zip:./ www3/secure.log	Sun Apr 30 2023 00:11:25 www3 sshd[5258]: Failed password for invalid user operator from 222.169.224.226 port 1711 ssh2
2023-04-29 00:19:01	tutorialdata.zip:./ www1/secure.log	Sat Apr 29 2023 00:19:01 www1 sshd[5258]: Failed password for invalid user rightscale from 87.194.216.51 port 3361 ssh2
2023-04-29 00:13:45	tutorialdata.zip:./ mailsv/secure.log	Sat Apr 29 2023 00:13:45 mailsv1 sshd[5258]: Failed password for invalid user testuser from 194.8.74.23 port 3626 ssh2
2023-04-28 00:23:28	tutorialdata.zip:./ www1/secure.log	Fri Apr 28 2023 00:23:28 www1 sshd[5258]: Failed password for invalid user redmine from 91.208.184.24 port 3587 ssh2

You extend the search using the `latest` function.

```
sourcetype=secure invalid user "sshd[5258]" | table _time source _raw |  
stats latest(_raw)
```

—

The search returns the event with the `_time` value `2023-05-01 00:15:05`, which is the event with the most recent timestamp.

<code>_time</code>	<code>source</code>	<code>_raw</code>
2023-05-01 00:15:05	tutorialdata.zip:./ mailsv/secure.log	Mon May 01 2023 00:15:05 mailsv1 sshd[5258]: Failed password for invalid user tomcat from 67.170.226.218 port 1490 ssh2

latest_time(<value>)

Description

Returns the UNIX time of the chronologically latest-seen occurrence of a given field value.

Usage

You can use this function with the `mstats`, `stats`, and `tstats` commands.

This function processes field values as strings.

If you have metrics data, you can use `latest_time` function in conjunction with `earliest`, `latest`, and `earliest_time` functions to calculate the rate of increase for a counter.

Alternatively, you can use the `rate` function counter to do the same thing.

Basic example

The following search runs against metric data. It is designed to return the latest UNIX time values in the past 60 minutes for metrics with names that begin with `queue.`.

```
| mstats latest_time(_value) where index=_metrics metric_name=queue.*
BY metric_name span=1m
```



The results appear on the Statistics tab and look something like this:

_time	metric_name	latest_time(_value)
2023-12-18 09:39:00	queue.current_size	1702921140.000000
2023-12-18 09:38:00	queue.current_size_kb	1702921080.000000
2023-12-18 09:37:00	queue.largest_size	1702921020.000000
2023-12-18 09:36:00	queue.max_size_kb	1702921020.000000
2023-12-18 09:35:00	queue.smallest_size	1702920900.000000
2023-12-18 09:34:00	queue.current_size	1702920840.000000
2023-12-18 09:33:00	queue.current_size_kb	1702920780.000000
2023-12-18 09:32:00	queue.largest_size	1702920720.000000
2023-12-18 09:31:00	queue.max_size_kb	1702920660.000000
2023-12-18 09:30:00	queue.smallest_size	1702920600.000000

per_day(<value>)

Description

Returns the values in a field or eval expression for each day.

Usage

You can use this function with the `timechart` command.

Basic examples

The following example returns the values for the field `total` for each day.

```
... | timechart per_day(total)
```

—

The following example returns the results of the eval expression

```
eval(method="GET")) AS Views .
```

```
... | timechart per_day(eval(method="GET")) AS Views
```

—

Extended example

This example uses the sample dataset from [the Search Tutorial](#) but should work with any format of Apache Web access log. Download the data set from [this topic in the Search Tutorial](#) and follow the instructions to upload it to your Splunk deployment.

This search uses the `per_day()` function and eval expressions to determine how many times the web pages were viewed and how many times items were purchased. The results appear on the Statistics tab.

```
sourcetype=access_* | timechart per_day(eval(method="GET")) AS  
Views_day, per_day(eval(action="purchase")) AS Purchases
```

—

To determine the number of Views and Purchases for each hour, minute, or second you can add the other time functions to the search. For example:

```
sourcetype=access_* | timechart per_day(eval(method="GET")) AS  
Views_day, per_hour(eval(method="GET")) AS Views_hour,  
per_minute(eval(method="GET")) AS Views_minute,  
per_day(eval(action="purchase")) AS Purchases
```

—

The screenshot shows the 'New Search' interface in Splunk. The search query is: `sourcetype=access_* | timechart per_day(eval(method="GET")) AS Views_day, per_hour(eval(method="GET")) AS Views_hour, per_minute(eval(method="GET")) AS Views_minute, per_day(eval(action="purchase")) AS Purchases`. The results table has columns: `_time`, `Views_day`, `Views_hour`, `Views_minute`, and `Purchases`. A 'field format option' dialog is open over the `Views_day` column, showing settings for Number Formatting: Precision (0), Use Thousand Separators (Yes), Unit (empty), and Unit Position (After).

_time	Views_day	Views_hour	Views_minute	Purchases
2018-03-08	792	33.000000	0.550000	157.000000
2018-03-09	3,737	166.709222	3.506120	783.000000
2018-03-10	3,580			903.000000
2018-03-11	3,715			829.000000
2018-03-12	3,393			781.000000
2018-03-13	3,477			793.000000
2018-03-14	3,506			797.000000
2018-03-15	2,666			694.000000

Use the field format option to change the number formatting for the field values.

per_hour(<value>)

Description

Returns the values in a field or eval expression for each hour.

Usage

You can use this function with the `timechart` command.

Basic examples

The following example returns the values for the field `total` for each hour.

```
... | timechart per_hour(total)
```

—

The following example returns the the results of the eval expression

```
eval(method="POST")) AS Views .
```

```
... | timechart per_hour(eval(method="POST")) AS Views
```

—

per_minute(<value>)

Description

Returns the values in a field or eval expression for each minute.

Usage

You can use this function with the `timechart` command.

Basic examples

The following example returns the values for the field `total` for each minute.

```
... | timechart per_minute(total)
```

▢

The following example returns the the results of the eval expression

```
eval(method="GET")) AS Views .
```

```
... | timechart per_minute(eval(method="GET")) AS Views
```

▢

per_second(<value>)

Description

Returns the values in a field or eval expression for each second.

Usage

You can use this function with the `timechart` command.

Basic examples

The following example returns the values for the field `kb` for each second.

```
... | timechart per_second(kb)
```

▢

rate(<value>)

Description

Returns the per-second rate change of the value in a field. The `rate` function represents the following formula:

```
(latest(<value>) - earliest(<value>)) / (latest_time(<value>) -  
earliest_time(<value>))
```

The `rate` function also handles the largest value reset if there is at least one reset.

Usage

You can use this function with the `mstats`, `stats`, and `tstats` commands.

- Provides the per-second rate change for an accumulating [counter metric](#). Accumulating counter metrics report the total counter value since the last counter reset. See [Investigate counter metrics](#) in *Metrics*
- Requires the `earliest` and `latest` values of the field to be numerical, and the `earliest_time` and `latest_time` values to be different.
- Requires at least two metric data points in the search time range.
- Should be used to provide rate information about single, rather than multiple, counters.

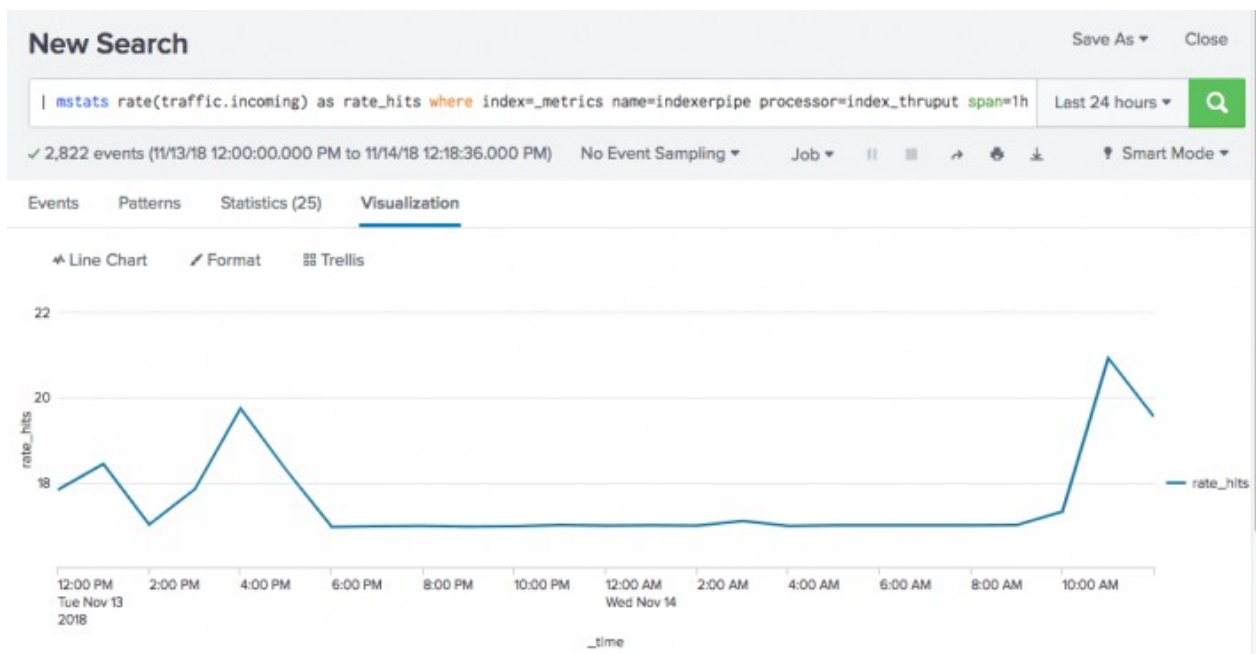
Basic example

The following search runs against metric data. It provides the hourly hit rate for a metric that provides measurements of incoming web traffic. It uses the `processor` filter to ensure that it is not reporting on multiple metric series (`name` and `processor` combinations).

```
| mstats rate(traffic.incoming) as rate_hits where index=_metrics
name=indexerpipe processor=index_thruput span=1h
```

○

The resulting chart shows you that the counter hit rate for the `traffic.incoming` metric spiked at 1 pm, 4 pm, and 11 am, but otherwise remained stable.



rate_avg(<value>)

Description

Computes the per [metric time series](#) rates for an accumulating [counter metric](#). Returns the averages of those rates.

For a detailed explanation of metric time series, see [Perform statistical calculations on metric time series](#) in *Metrics*.

Usage

You can use this function with the `mstats` command.

- To ensure accurate results, Splunk software uses the latest value of a metric measurement from the previous timespan as the starting basis for a rate computation.
- When you calculate the average rates for accumulating counter metrics, the cleanest way to do it is to split the counter metric rate calculations out by metric time series and then compute the average rate across all of the metric time series.
- Unlike `rate`, the `rate_avg` function can calculate rates even when there is only a single metric data point per time series per timespan. It can pull in data across timespans to calculate rates when necessary.
- The `rate_avg` function does not support `prestats=true`. It needs the final list of dimensions to split by.

Basic example

In your `_metrics` index, you have data for the metric

`spl.intr.resource_usage.PerProcess.data.elapsed`. This is an accumulating counter metric. It contains a number of metric time series.

The following example search uses the `rate_avg` function to calculate the `rate(X)` for each `spl.mlog.thruput.thruput.total_k_processed` time series in the time range. Then it gets the average rate across all of the time series. Lastly, it splits the results by time, so they can be plotted on a chart.

```
| mstats rate_avg(spl.mlog.thruput.thruput.total_k_processed) where
index=_metrics span=1h
```

rate_sum(<value>)

Description

Computes the per [metric time series](#) rates for an accumulating [counter metric](#). Returns the aggregate of those rates.

For a detailed explanation of metric time series, see [Perform statistical calculations on metric time series](#) in *Metrics*.

Usage

You can use this function with the `mstats` command.

- To ensure accurate results, Splunk software uses the latest value of a metric measurement from the previous timespan as the starting basis for a rate computation.
- When you calculate the aggregated rates for accumulating counter metrics, the cleanest way to do it is to split the counter metric rate calculations out by metric time series and then compute the aggregate rate across all of the metric time series.
- Unlike `rate`, the `rate_sum` function can calculate rates even when there is only a single metric data point per time series per timespan. It can pull in data across timespans to calculate rates when necessary.
- The `rate_sum` function does not support `prestats=true`. It needs the final list of dimensions to split by.

Basic example

In your `_metrics` index, you have data for the metric

`spl.intr.resource_usage.PerProcess.data.elapsed`. This is an accumulating counter metric. It contains a number of metric time series.

The following example search uses the `rate_sum` function to calculate the `rate(X)` for each `spl.mlog.thruput.thruput.total_k_processed` time series in the time range. Then it gets the aggregate rate across all of the time series. Lastly, it splits the results by time, so they can be plotted on a chart.

```
| mstats rate_sum(spl.mlog.thruput.thruput.total_k_processed) where
index=_metrics span=1h
```

