# Explaining Value Interpolation with PCP Archives


Ken McDonell
kenj@kenj.id.au

Version 1.1
Dec 2023

As part of the Performance Co-Pilot (PCP) historical records of performance data are maintained as time-series of observations in a PCP archive. In the first instance these archives are created by **pmlogger**(1).

This document aims to explain how when a PCP archive is replayed values are instantiated for a metric when fetched at an arbitrary point of time within the archive.

## *Background*

The PCP architecture offers a number of differentiators compared to other performance data collection infrastructures, namely:

1. Real-time sources and historical sources (PCP archives) are largely interchangeable, meaning the same reporting, display, alerting and analysis tools can be used for both real-time monitoring and historical analysis.
2. In addition to the data values associated with a performance metric, there is metadata that describes the syntax and semantics of each performance metric. This metadata can be fetched from real-time sources and is stored in the PCP archives.
3. A metric may have a singular value, e.g. proc.nproc (the number of running processes) or may have a set-value, e.g. proc.psinfo.utime (the user-mode CPU time for every running process). Where a metric is set-valued, the associated set of instances at any point in time is defined by a metric's instance domain. Multiple metrics may share the same instance domain. Instance domains can be queried from real-time sources and are stored as a time-series in the PCP archives.
4. Groups of metrics in PCP archives may have different logging intervals. Some, like hardware configuration, may be logged once. Others that are subject to slow change may be logged infrequently, e.g. the 15-minute load average. And others may require more frequent logging, e.g. disk or network activity.
5. PCP archives can be sliced-diced-and-merged to create a new archive with subsets of the metrics, parts of the time series, concatenation of time series, changes in the sampling interval, etc.
6. When replaying performance data from a PCP archive, a monitoring application will typically want values for a number of metrics, at a specific a sampling interval (and possibly a starting time), so the "time" for each sample of the replay is not correlated with the "time" for observations recorded

in the PCP archive.

In the remainder of this document we shall largely disregard set-valued metrics and instance domains. All of the principles and logic that is outlined below relating to value interpolation from PCP archives for a singular metric may be applied to the value for each instance of a set-valued metric.

## *Metadata and Data Semantics*

The PCP metadata describes each metric and includes:

- unique metric identifier (pmID)
- data type (integers of various types and sizes, floats, doubles, strings, etc.)
- data semantics (counter, instantaneous, discrete)
- data scale (dimension and scale for time x space x count)
- instance domain identification (pmInDom)
- help text
- labels

The -d (core metadata), -l (metric labels), -t (one line help text) and -T (verbose help text) options to **pminfo**(1) will report the metadata for one or more metrics.

When instantiating values for a metric from an archive it is the **data semantics** that is most important.

### *Counter Semantics*

Metrics with COUNTER semantics have values that are monotonic increasing over time. By definition they have a numeric type. These metrics are typically reset to zero at boot time, process creation or service startup, and thereafter are free running.

Examples: disk.dev.read_bytes (number of Kbytes read for each disk device), swap.pagesout (number of pages written to the swap device), proc.psinfo.stime (msec of system CPU time for each process).

It is important to note that most applications that consume PCP data will convert metrics with COUNTER semantics into rates by sampling the metric twice then dividing the difference in value by the difference in time. So disk.dev.read_bytes will be reported as Kbytes/sec, swap.pagesout will be reported as count/sec and proc.psinfo.stime will be reported as msec/msec (or utilization of one CPU).

### *Instantaneous Semantics*

Metrics with INSTANTANEOUS semantics have values that may increase, or decrease, or not change, between one observation and the next.

Examples: disk.dev.util (percentage of time each disk was busy processing requests), swap.used (total swap used in units of bytes), proc.psinfo.cgroups (text string encoding the list of process cgroups for each process).

### *Discrete Semantics*

Similar to INSTANTANEOUS semantics, DISCRETE semantics are those that are unlikely to change from one observation to the next. Most metrics with DISCRETE semantics have values that report system sizes or configuration parameters.

Examples: disk.partitions.capacity (size of each disk partition, in units of Kbytes), pmcd.timezone (timezone for the **pmcd**(1) process), hinv.physmem (installed RAM, in units of Mbytes), proc.psinfo.start_time (start time for each process relative to system boot time, in units of msec).

## *Interpolation Algorithms*

For some random singular metric we shall use the notation xa(t) to mean the value that is stored in the PCP archive at some sample time t, and xf(t) to mean the value that is returned from **pmFetch**(3) at

some arbitrary time t when the archive is replayed.

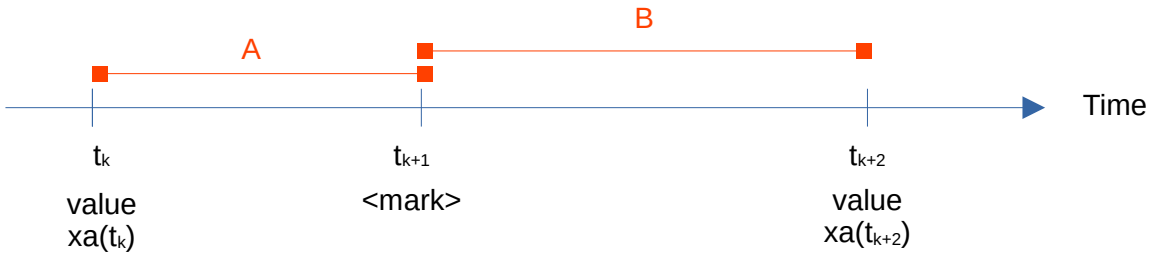
## *Counter Semantics*

The value for a metric with COUNTER semantics at an arbitrary time t is:

- If there is some $xa(t_l)$ for a largest $t_l \leq t$, then this is the prior bound.
- If there is some $xa(t_u)$ for a smallest $t_u \geq t$, then this is the next bound.
- If both the prior bound and the next bound exist, then xf(t) is computed by linear interpolation, i.e. $xf(t) = xa(t_l) + (t-t_l) * (xa(t_u) - xa(t_l)) / (t_u - t_l)$, otherwise no value exists[1] for xf(t).

The bounds may not exist for the following reasons:

- For all times before the first observation of a metric in the archive, there is no prior bound.
- For all times after the last observation of a metric in the archive, there is no next bound.
- Discontinuities in a PCP archive are indicated by a <mark> record; this can occur when a system is reboot, or archive logging is suspended for a time, or archives from disjoint time periods are combined with **pmlogextract**(1). The <mark> record has a timestamp. As shown in the diagram below, there is no next bound in the region A (after $t_k$ and up to $t_{k+1}$). Similarly there is no prior bound in the region B (from $t_{k+1}$ up to $t_{k+2}$). So no values are available across both regions A and B.




## *Instantaneous Semantics*

There is no way to determine exact values between observations of a metric with INSTANTANEOUS semantics, but provided the time between the observations is small (in comparison to the likely rate of change of the value), then the “last” observed value is a reasonable approximation.

This corresponds with the semantics associated with **sar**(1) or **vmstat**(1) when monitoring free memory for example.

The value for a metric with INSTANTANEOUS semantics at an arbitrary time t is:

- If there is some $xa(t_l)$ for a largest $t_l \leq t$, then this is the prior bound.
- If there is some $xa(t_u)$ for a smallest $t_u \geq t$, then this is the next bound.
- If both the prior bound and the next bound exist, then xf(t) is computed by extrapolation, i.e. $xf(t) = xa(t_l)$, otherwise no value exists for xf(t).

The conditions for the bounds not existing are identical to those described for a metric with COUNTER semantics.

## *Discrete Semantics*

This is the same as the INSTANTANEOUS case, except the next bound requirement is removed.

1 The PCP protocols support encoding “No value is available” for a particular metric at a particular time in the result returned from **pmFetch**(3).

The value for a metric with DISCRETE semantics at an arbitrary time t is:

- If there is some xa($t_i$) for a largest $t_i \leq t$, then this is the prior bound.
- If the prior bound exists, then xf(t) is computed by extrapolation, i.e. xf(t) = xa($t_i$), otherwise no value exists for xf(t).

## *Annotated Examples*

We shall consider some very simple examples, with the following constraints.

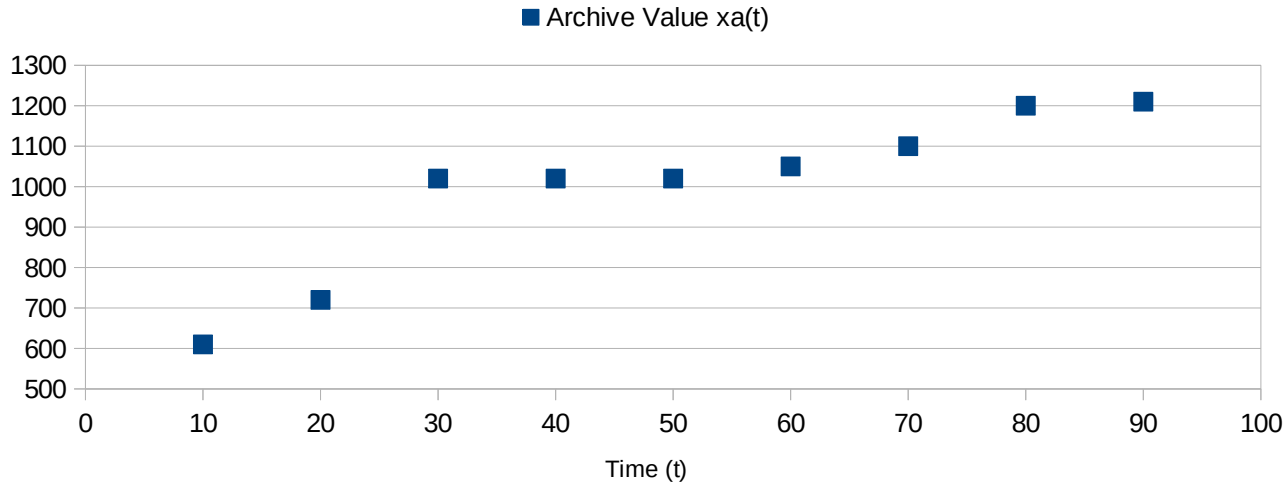
- We show only singular metrics, e.g. disk.total.read. For the more general set-valued metrics (those with an Instance Domain in PCP terminology), the same algorithms and calculations are made independently for **every** instance of the metric. Alternatively, consider x(t) to be the value of a single metric-instance pair at time t, e.g. the value for the “sda” instance of the metric disk.dev.read.
- The values in the archive are shown at regular times. In practice, different metrics are logged at different intervals, while the effects of the configurations used with **pmlogger**(1) to create an archive and run-time reconfiguration with **pmlc**(1) followed by archive slice-n-dice operations with **pmlogmerge**(1) or **pmlogextract**(1) all mean that the sampling interval for a single value is not constant over an archive.
- The archive timestamps (t) have been converted into seconds relative to the start of the archive, and t is in the range 0 to 99.
- There are no discontinuities or <mark> records as discussed above.

These constraints limit the complexity of the examples in the hope that the base concepts will be more easily understood.

### *Counter Semantics*

Consider the following values for a singular metric (x) with COUNTER semantics as recorded in a PCP archive over time. As described above, we will use the notation xa(t) to mean the value of x stored in the archive at time t.

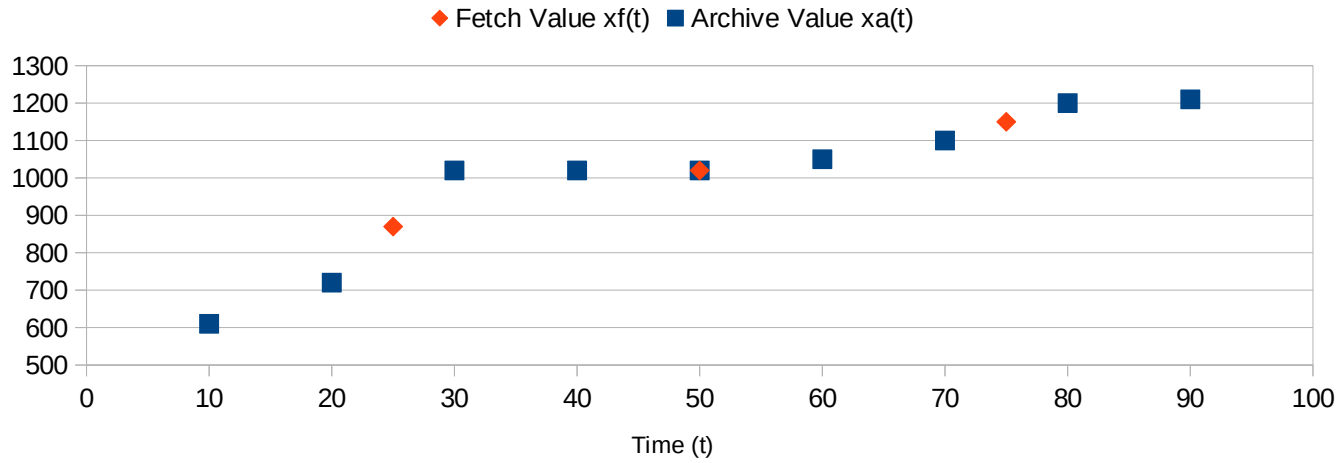
| Time (t) | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |
|---|---|---|---|---|---|---|---|---|---|
| Archive Value xa(t) | 610 | 720 | 1020 | 1020 | 1020 | 1050 | 1100 | 1200 | 1210 |



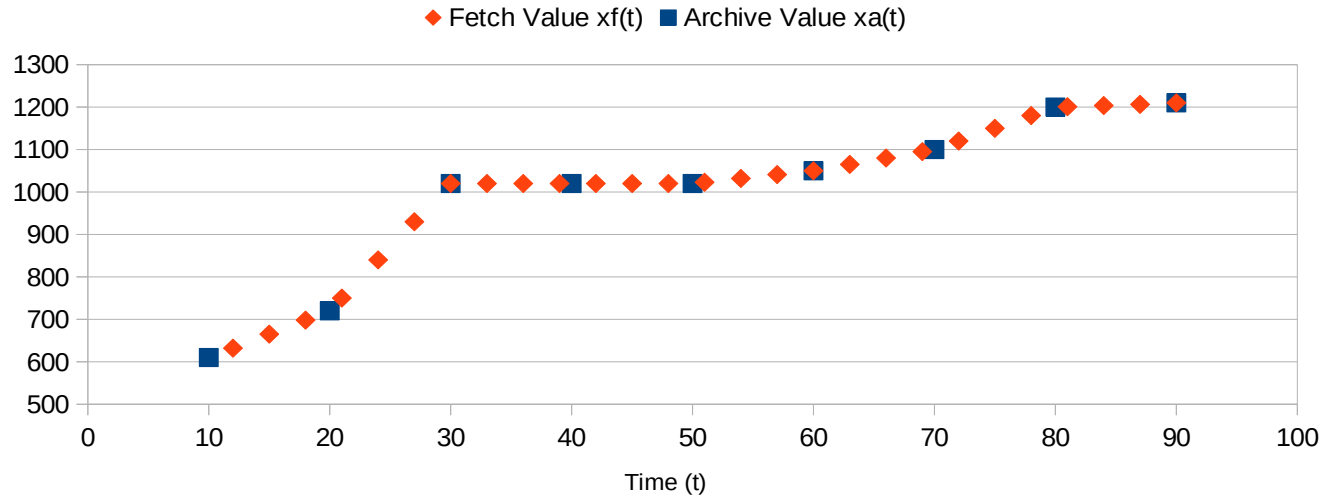

Now this archive is to be replayed to report the values of x that are returned by **pmFetch**(3) at time t,

designated xf(t). If the interval between fetches was 25 and we start at the beginning of the archive, then the values would be as shown in red on the graph below, computed by linear interpolation between the recorded data values (shown in blue) immediately before and after the requested times. So xf(25) = 870, xf(50) = 1020 (requested time equals a recorded time, so use that value) and xf(75) = 1150.

| Time (t) | 10 | 20 | 25 | 30 | 40 | 50 | 60 | 70 | 75 | 80 | 90 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Archive Value xa(t) | 610 | 720 | | 1020 | 1020 | 1020 | 1050 | 1100 | | 1200 | 1210 |
| Fetch Value xf(t) | | | 870 | | | 1020 | | | 1150 | | |




In the example above, the replay interval (25) is larger than the logging interval (10). But the opposite may apply, so in the example below the replay interval (3) is less than the logging interval (10).

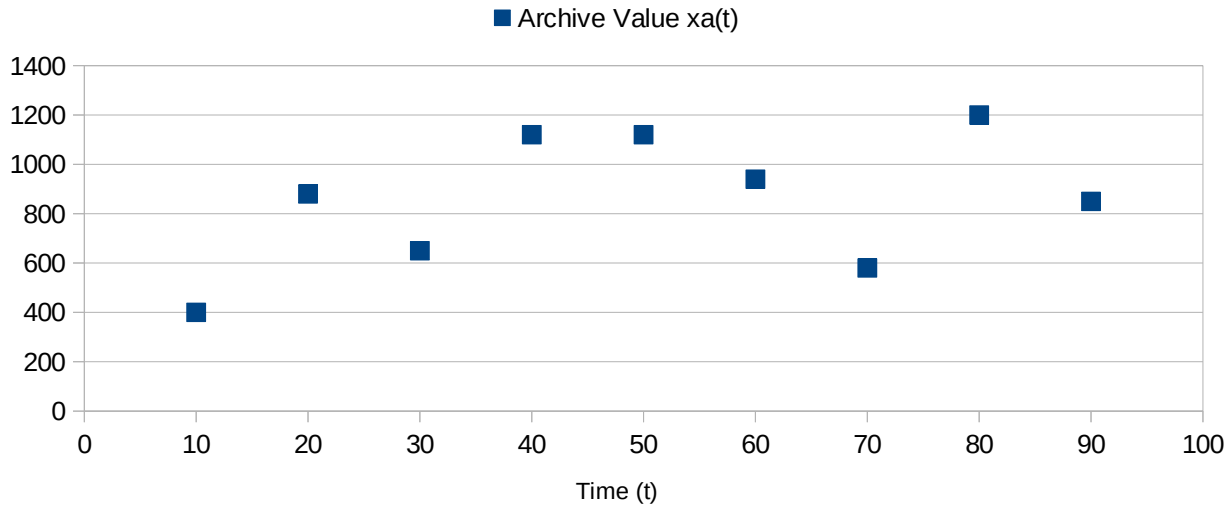



In both cases, note that there are no values for xf(t) when t < 10 because there is no prior bound. Similarly there are no values for xf(t) when t > 90 because there is no next bound.

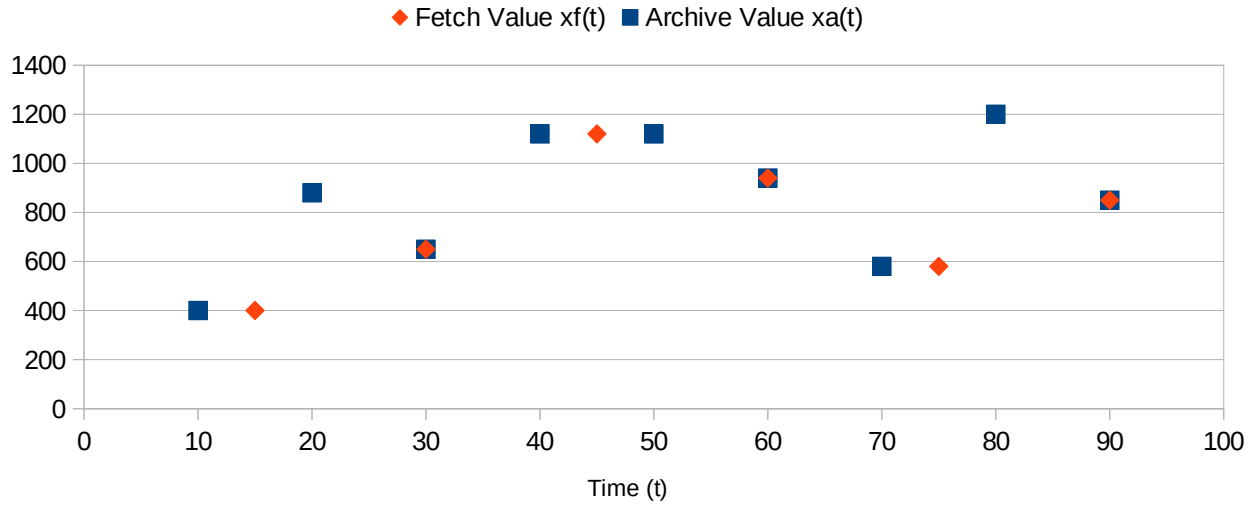
### *Instantaneous Semantics*

Consider the following values for a singular metric (x) with INSTANTANEOUS semantics as recorded in a PCP archive over time. As described above, we will use the notation xa(t) to mean the value of x stored in the archive at time t.

| Time (t) | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |
|---|---|---|---|---|---|---|---|---|---|
| Archive Value xa(t) | 400 | 880 | 650 | 1120 | 1120 | 940 | 580 | 1200 | 850 |



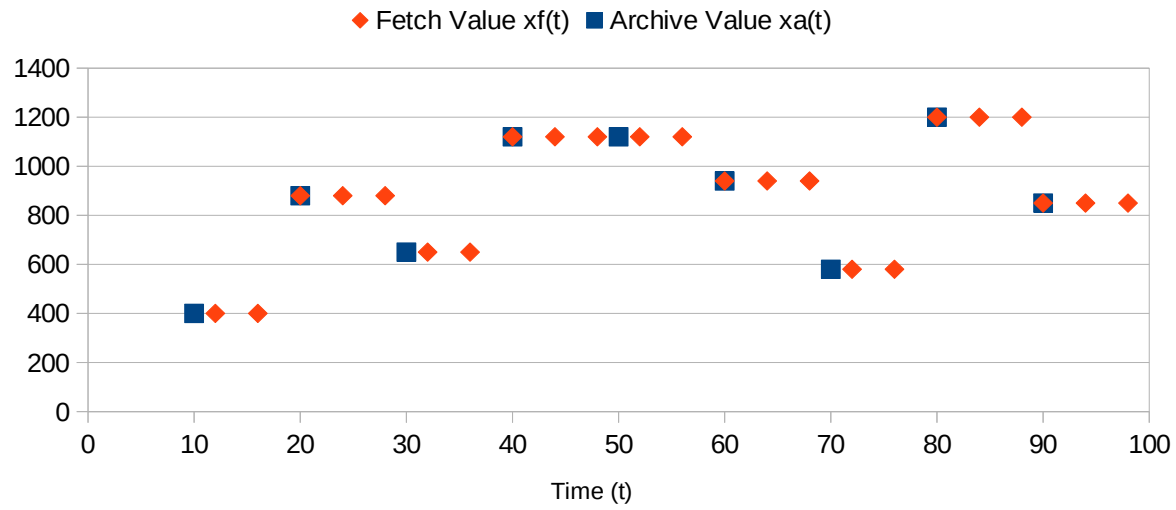

Now this archive is to be replayed to report the values of x that are returned by **pmFetch**(3) at time t, designated xf(t). If the interval between fetches was 15 and we start at the beginning of the archive, then the values would be as shown in red on the graph below.




### *Discrete Semantics*

Using the same data as in the previous example, but changing the semantics to DISCRETE and changing the interval between fetches to 4, would produce the results shown below.

Note that in this case the values are returned for the times t > 90, where no next bound exists.




## *Issues Adding Extra Complexity*

Set-valued metrics are more complicated because:

- not all instances are necessarily logged at the same time in the archive
- some instances may appear and others may disappear during the period spanned by the archive
- the client may request only a subset of the instances in a particular **pmFetch**(3)

Replay may proceed in either a forwards or a backwards direction, and the direction can be changed by the monitoring application.

Replay may commence an arbitrary point within the time period spanned by the archive.

Multiple <mark> records may be present in a single archive.

Metrics with COUNTER semantics may be reset (hardware disabled and re-enabled, or some service stopped and re-started) or overflow or risk arithmetic overflow in the interpolation calculation.

When all these possible cases are combined, the process of establishing the next and prior bounds for a specific metric-instance value may involve scanning the entire archive both forwards and backwards. The implementation tries very hard to gather as much state as possible during each of these scans to minimize the number of archive records read (and possibly decompressed) during scanning.

Fortunately, for most metrics and most archives, the most common value instantiation avoids the complex corner cases and follows the algorithms described above with simple incremental reading of the archive as replay progresses.

## *May the Source be With You*

The definitive truth about value instantiation is in the *libpcp* source code, specifically in the 2000+ lines of C source that may be inspected at https://github.com/performancecopilot/pcp/blob/main/src/libpcp/src/interp.c

Good luck!