## NAME
rrdgraph_rpn − About RPN Math in rrdtool graph

## SYNOPSIS
*RPN expression*:=*vname*|*operator*|*value*[,*RPN expression*]

## DESCRIPTION
If you have ever used a traditional HP calculator you already know **RPN** (Reverse Polish Notation). The idea behind **RPN** is that you have a stack and push your data onto this stack. Whenever you execute an operation, it takes as many elements from the stack as needed. Pushing is done implicitly, so whenever you specify a number or a variable, it gets pushed onto the stack automatically.

At the end of the calculation there should be one and only one value left on the stack. This is the outcome of the function and this is what is put into the *vname*. For **CDEF** instructions, the stack is processed for each data point on the graph. **VDEF** instructions work on an entire data set in one run. Note, that currently **VDEF** instructions only support a limited list of functions.

Example: VDEF:`maximum=mydata,MAXIMUM`

This will set variable "maximum" which you now can use in the rest of your RRD script.

Example: CDEF:`mydatabits=mydata,8,*`

This means: push variable *mydata*, push the number 8, execute the operator *\**. The operator needs two elements and uses those to return one value. This value is then stored in *mydatabits*. As you may have guessed, this instruction means nothing more than *mydatabits = mydata \* 8*. The real power of **RPN** lies in the fact that it is always clear in which order to process the input. For expressions like `a = b + 3 * 5` you need to multiply 3 with 5 first before you add *b* to get *a*. However, with parentheses you could change this order: `a = (b + 3) * 5`. In **RPN**, you would do `a = b, 3, +, 5, *` without the need for parentheses.

## OPERATORS
Boolean operators
### LT, LE, GT, GE, EQ, NE

Less than, Less or equal, Greater than, Greater or equal, Equal, Not equal all pop two elements from the stack, compare them for the selected condition and return 1 for true or 0 for false. Comparing an *unknown* or an *infinite* value will result in *unknown* returned ... which will also be treated as false by the **IF** call.

### UN, ISINF

Pop one element from the stack, compare this to *unknown* respectively to *positive or negative infinity*. Returns 1 for true or 0 for false.

*then*,*else*,*condition*,**IF**

Pops three elements from the stack. If the element popped last is 0 (false), the value popped first is pushed back onto the stack, otherwise the value popped second is pushed back. This does, indeed, mean that any value other than 0 is considered to be true.

Example: `A,B,C,IF` should be read as `if (A) then (B) else (C)`

Comparing values
### MIN, MAX

Pops two elements from the stack and returns the smaller or larger, respectively. Note that *infinite* is larger than anything else. If one of the input numbers is *unknown* then the result of the operation will be *unknown* too.

### MINNAN, MAXNAN

NAN-safe version of MIN and MAX. If one of the input numbers is *unknown* then the result of the

operation will be the other one. If both are *unknown*, then the result of the operation is *unknown*.

*lower-limit*,*upper-limit*,**LIMIT**

Pops two elements from the stack and uses them to define a range. Then it pops another element and if it falls inside the range, it is pushed back. If not, an *unknown* is pushed.

The range defined includes the two boundaries (so: a number equal to one of the boundaries will be pushed back). If any of the three numbers involved is either *unknown* or *infinite* this function will always return an *unknown*

Example: `CDEF:a=alpha,0,100,LIMIT` will return *unknown* if alpha is lower than 0 or if it is higher than 100.

Arithmetics
   **+, −, \*, /, %**

   Add, subtract, multiply, divide, modulo

   **ADDNAN**

   NAN-safe addition. If one parameter is NAN/UNKNOWN it'll be treated as zero. If both parameters are NAN/UNKNOWN, NAN/UNKNOWN will be returned.

   *value*,*power*,**POW**

   Raise *value* to the power of *power*.

   **SIN, COS, LOG, EXP, SQRT**

   Sine and cosine (input in radians), log and exp (natural logarithm), square root.

   **ATAN**

   Arctangent (output in radians).

   **ATAN2**

   Arctangent of y,x components (output in radians). This pops one element from the stack, the x (cosine) component, and then a second, which is the y (sine) component. It then pushes the arctangent of their ratio, resolving the ambiguity between quadrants.

   Example: `CDEF:angle=Y,X,ATAN2,RAD2DEG` will convert `X,Y` components into an angle in degrees.

   **FLOOR, CEIL**

   Round down or up to the nearest integer.

   **DEG2RAD, RAD2DEG**

   Convert angle in degrees to radians, or radians to degrees.

   **ABS**

   Take the absolute value.

Set Operations
   *count*,**SORT**

   Pop one element from the stack. This is the *count* of items to be sorted. The top *count* of the remaining elements are then sorted from the smallest to the largest, in place on the stack.

       4,3,22.1,1,4,SORT -> 1,3,4,22.1

   *count*,**REV**

Reverse the number

Example:   `CDEF:x=v1,v2,v3,v4,v5,v6,6,SORT,POP,5,REV,POP,+,+,+,4,/`   will compute the average of the values v1 to v6 after removing the smallest and largest.

*count*,**AVG**

Pop one element (*count*) from the stack. Now pop *count* elements and build the average, ignoring all UNKNOWN values in the process.

Example: `CDEF:x=a,b,c,d,4,AVG`

*count*,**SMIN** and *count*,**SMAX**

Pop one element (*count*) from the stack. Now pop *count* elements and push the minimum/maximum back onto the stack.

Example: `CDEF:x=a,b,c,d,4,AVG`

*count*,**MEDIAN**

pop one element (*count*) from the stack. Now pop *count* elements and find the median, ignoring all UNKNOWN values in the process. If there are an even number of non-UNKNOWN values, the average of the middle two will be pushed on the stack.

Example: `CDEF:x=a,b,c,d,4,MEDIAN`

*count*,**STDEV**

pop one element (*count*) from the stack. Now pop *count* elements and calculate the standard deviation over these values (ignoring any NAN values). Push the result back on to the stack.

Example: `CDEF:x=a,b,c,d,4,STDEV`

*percent*,*count*,**PERCENT**

pop two elements (*count*,*percent*) from the stack. Now pop *count* element, order them by size (while the smalles elements are −INF, the largest are INF and NaN is larger than −INF but smaller than anything else. No pick the element from the ordered list where *percent* of the elements are equal then the one picked. Push the result back on to the stack.
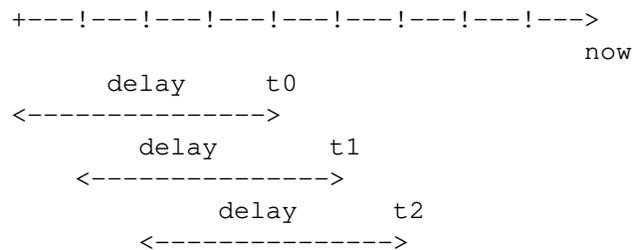
Example: `CDEF:x=a,b,c,d,95,4,PERCENT`

*count*,**TREND, TRENDNAN**

Create a ''sliding window'' average of another data series.

Usage: CDEF:smoothed=x,1800,TREND

This will create a half-hour (1800 second) sliding window average of x. The average is essentially computed as shown here:

```
            +---!---!---!---!---!---!---!---!--->
                                            now

              delay       t0
          <--------------->
                delay        t1
            <--------------->
                    delay       t2
              <--------------->


    Value at sample (t0) will be the average between (t0-delay) and (t0)
    Value at sample (t1) will be the average between (t1-delay) and (t1)
    Value at sample (t2) will be the average between (t2-delay) and (t2)
```

TRENDNAN is − in contrast to TREND − NAN-safe. If you use TREND and one source value is NAN the complete sliding window is affected. The TRENDNAN operation ignores all NAN-values in a sliding window and computes the average of the remaining values.
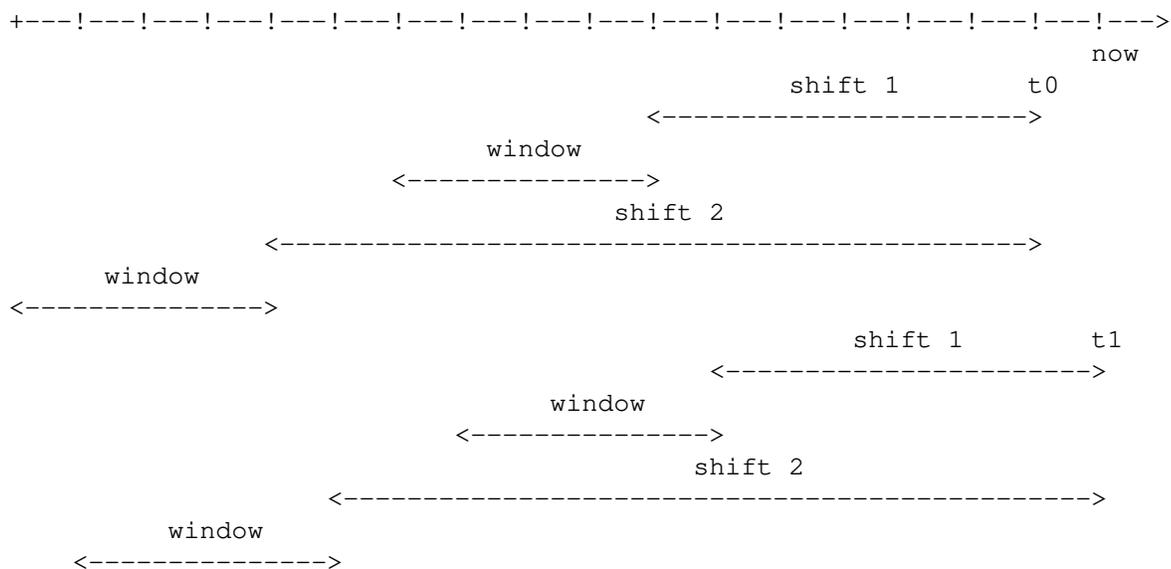
**PREDICT, PREDICTSIGMA, PREDICTPERC**

Create a ''sliding window'' average/sigma/percentil of another data series, that also shifts the data series by given amounts of time as well

Usage – explicit stating shifts: `CDEF:predict=<shift n>,...,<shift 1>,n,<window>,x,PREDICT` `CDEF:sigma=<shift n>,...,<shift 1>,n,<window>,x,PREDICTSIGMA` `CDEF:perc=<shift n>,...,<shift 1>,n,<window>,<percentil>,x,PREDICTPERC`

Usage – shifts defined as a base shift and a number of time this is applied `CDEF:predict=<shift multiplier>,−n,<window>,x,PREDICT` `CDEF:sigma=<shift multiplier>,−n,<window>,x,PREDICTSIGMA` `CDEF:sigma=<shift multiplier>,−n,<window>,<percentil>,x,PREDICTPERC`

Example: CDEF:predict=172800,86400,2,1800,x,PREDICT

This will create a half-hour (1800 second) sliding window average/sigma of x, that average is essentially computed as shown here:

```
 +---!---!---!---!---!---!---!---!---!---!---!---!---!---!---!---!---!--->
                                                                    now
                                          shift 1            t0
                                       <---------------------->
                           window
                       <--------------->
                                            shift 2
                   <----------------------------------------------->
           window
   <--------------->

                                            shift 1           t1
                                         <---------------------->
                       window
                   <--------------->
                                          shift 2
                   <----------------------------------------------->
           window
       <--------------->
```

```
 Value at sample (t0) will be the average between (t0-shift1-window) and (t0-shif
                                   and between (t0-shift2-window) and (t0-shif
 Value at sample (t1) will be the average between (t1-shift1-window) and (t1-shif
                                   and between (t1-shift2-window) and (t1-shif
```

The function is by design NAN-safe. This also allows for extrapolation into the future (say a few days) − you may need to define the data series with the optional start= parameter, so that the source data series has enough data to provide prediction also at the beginning of a graph...

The percentile can be between [−100:+100]. The positive percentiles interpolates between values while the negative will take the closest.

Example: you run 7 shifts with a window of 1800 seconds. Assuming that the rrd-file has a step size of 300 seconds this means we have to do the percentile calculation based on a max of 42 distinct values (less if you got NAN). that means that in the best case you get a step rate between values of 2.4 percent. so if you ask for the 99th percentile, then you would need to look at the 41.59th value. As we

only have integers, either the 41st or the 42nd value.

With the positive percentile a linear interpolation between the 2 values is done to get the effective value.

The negative returns the closest value distance wise − so in the above case 42nd value, which is effectively returning the Percentile100 or the max of the previous 7 days in the window.

Here an example, that will create a 10 day graph that also shows the prediction 3 days into the future with its uncertainty value (as defined by avg+−4*sigma) This also shows if the prediction is exceeded at a certain point.

```
rrdtool graph image.png --imgformat=PNG \
--start=-7days --end=+3days --width=1000 --height=200 --alt-autoscale-max \
DEF:value=value.rrd:value:AVERAGE:start=-14days \
LINE1:value#ff0000:value \
CDEF:predict=86400,-7,1800,value,PREDICT \
CDEF:sigma=86400,-7,1800,value,PREDICTSIGMA \
CDEF:upper=predict,sigma,3,*,+ \
CDEF:lower=predict,sigma,3,*,- \
LINE1:predict#00ff00:prediction \
LINE1:upper#0000ff:upper\ certainty\ limit \
LINE1:lower#0000ff:lower\ certainty\ limit \
CDEF:exceeds=value,UN,0,value,lower,upper,LIMIT,UN,IF \
TICK:exceeds#aa000080:1 \
CDEF:perc95=86400,-7,1800,95,value,PREDICTPERC \
LINE1:perc95#ffff00:95th_percentile
```

Note: Experience has shown that a factor between 3 and 5 to scale sigma is a good discriminator to detect abnormal behavior. This obviously depends also on the type of data and how ''noisy'' the data series is.

Also Note the explicit use of start= in the CDEF − this is necessary to load all the necessary data (even if it is not displayed)

This prediction can only be used for short term extrapolations − say a few days into the future.

Special values

**UNKN**

Pushes an unknown value on the stack

**INF, NEGINF**

Pushes a positive or negative infinite value on the stack. When such a value is graphed, it appears at the top or bottom of the graph, no matter what the actual value on the y−axis is.

**PREV**

Pushes an *unknown* value if this is the first value of a data set or otherwise the result of this **CDEF** at the previous time step. This allows you to do calculations across the data. This function cannot be used in **VDEF** instructions.

**PREV(vname)**

Pushes an *unknown* value if this is the first value of a data set or otherwise the result of the vname variable at the previous time step. This allows you to do calculations across the data. This function cannot be used in **VDEF** instructions.

**COUNT**

Pushes the number 1 if this is the first value of the data set, the number 2 if it is the second, and so on. This special value allows you to make calculations based on the position of the value within the data

set. This function cannot be used in **VDEF** instructions.

Time

Time inside RRDtool is measured in seconds since the epoch. The epoch is defined to be `Thu Jan  1 00:00:00 UTC 1970`.

**NOW**

Pushes the current time on the stack.

**STEPWIDTH**

The width of the current step in seconds. You can use this to go back from rate based presentations to absolute numbers

    CDEF:abs=rate,STEPWIDTH,*,PREV,ADDNAN

**NEWDAY,NEWWEEK,NEWMONTH,NEWYEAR**

These three operators will return 1.0 whenever a step is the first of the given period. The periods are determined according to the local timezone AND the `LC_TIME` settings.

    CDEF:mtotal=rate,STEPWIDTH,*,NEWMONTH,0,PREV,IF,ADDNAN

**TIME**

Pushes the time the currently processed value was taken at onto the stack.

**LTIME**

Takes the time as defined by **TIME**, applies the time zone offset valid at that time including daylight saving time if your OS supports it, and pushes the result on the stack.  There is an elaborate example in the examples section below on how to use this.

Processing the stack directly

**DUP, POP, EXC**

Duplicate the top element, remove the top element, exchange the two top elements.

**DEPTH**

pushes the current depth of the stack onto the stack

    a,b,DEPTH -> a,b,2

n,**COPY**

push a copy of the top n elements onto the stack

    a,b,c,d,2,COPY => a,b,c,d,c,d

n,**INDEX**

push the nth element onto the stack.

    a,b,c,d,3,INDEX -> a,b,c,d,b

n,m,**ROLL**

rotate the top n elements of the stack by m

    a,b,c,d,3,1,ROLL => a,d,b,c
    a,b,c,d,3,-1,ROLL => a,c,d,b

## VARIABLES

These operators work only on **VDEF** statements. Note that currently ONLY these work for **VDEF**.

MAXIMUM, MINIMUM, AVERAGE
Return the corresponding value, MAXIMUM and MINIMUM also return the first occurrence of that value in the time component.

Example: VDEF:avg=mydata,AVERAGE

STDEV
Returns the standard deviation of the values.

Example: VDEF:stdev=mydata,STDEV

LAST, FIRST
Return the last/first non-nan or infinite value for the selected data stream, including its timestamp.

Example: VDEF:first=mydata,FIRST

TOTAL
Returns the rate from each defined time slot multiplied with the step size. This can, for instance, return total bytes transferred when you have logged bytes per second. The time component returns the number of seconds.

Example: VDEF:total=mydata,TOTAL

PERCENT, PERCENTNAN
This should follow a **DEF** or **CDEF** *vname*. The *vname* is popped, another number is popped which is a certain percentage (0..100). The data set is then sorted and the value returned is chosen such that *percentage* percent of the values is lower or equal than the result. For PERCENTNAN *Unknown* values are ignored, but for PERCENT *Unknown* values are considered lower than any finite number for this purpose so if this operator returns an *unknown* you have quite a lot of them in your data. **Inf**inite numbers are lesser, or more, than the finite numbers and are always more than the *Unknown* numbers. (NaN < −INF < finite values < INF)

Example: VDEF:perc95=mydata,95,PERCENT
    VDEF:percnan95=mydata,95,PERCENTNAN

LSLSLOPE, LSLINT, LSLCORREL
Return the parameters for a **L**east **S**quares **L**ine *(y = mx +b)* which approximate the provided dataset. LSLSLOPE is the slope *(m)* of the line related to the COUNT position of the data. LSLINT is the y−intercept *(b)*, which happens also to be the first data point on the graph. LSLCORREL is the Correlation Coefficient (also know as Pearson's Product Moment Correlation Coefficient). It will range from 0 to +/−1 and represents the quality of fit for the approximation.

Example: VDEF:slope=mydata,LSLSLOPE

# SEE ALSO

rrdgraph gives an overview of how **rrdtool graph** works. rrdgraph_data describes **DEF**,**CDEF** and **VDEF** in detail. rrdgraph_rpn describes the **RPN** language used in the **?DEF** statements. rrdgraph_graph page describes all of the graph and print functions.

Make sure to read rrdgraph_examples for tips&tricks.

# AUTHOR

Program by Tobias Oetiker <tobi@oetiker.ch>

This manual page by Alex van den Bogaerdt <alex@vandenbogaerdt.nl> with corrections and/or additions by several people