# Baseline NodeBrain Module

**Release 0.8.17**

Baseline NodeBrain Module
August 2014
NodeBrain Open Source Project

**Release 0.8.17**

Author: Ed Trettevik

**History**

| | |
|---|---|
| 2010-09-13 | Title: *Baseline NodeBrain Module*<br>Author: Ed Trettevik <eat@nodebrain.org><br>Publisher: NodeBrain Open Source Project<br><br>Version 0.1<br>• Initial prototype. This document describes the Baseline module as first introduced with NodeBrain 0.8.3, with very little experimentation on real world problems having been performed to validate the design. We anticipate changes to this document as the module evolves, hopefully quickly, to a version 1.0. |

## Preface

This manual is intended for users of the Baseline NodeBrain Module, a plug-in for statistical anomaly detection. The Baseline module was first introduced in NodeBrain 0.8.3 in September 2010. This module should be treated as a prototype. It has not yet been exercised enough to fully validate the design. We expect it to evolve as we gain experience.

A statistical anomaly detection capability has been on the NodeBrain to-do list since 2003. Although theoretically interesting, and potentially useful, we did not consider it a high priority. Around 2008, it moved into the "management requested" category, but continued at a lower priority than other projects consuming development resources. In August 2010, disruption on another project requiring a temporary halt for redesign and rescheduling provided a nice opportunity to work on this task until the dust settled on the higher priority project. A several year old draft design was reviewed and then mostly discarded and replaced by a very simple approach based on the existing Tree module. This made it possible to create the initial prototype in a couple days.

## Documents

*NodeBrain Guide* - Information on using `nb`
*NodeBrain Tutorial* - A gentle introduction to `nb` and the rule language
*NodeBrain Language* - Rule language syntax and semantics
*NodeBrain Library* - C API

## Document Conventions

Sample code and input/output examples are displayed in a monospace font, indented in HTML and Info, and enclosed in a box in PDF or printed copies. Bold text is used to bring the reader's attention to specific portions of an example. In the following example, the first and last line are associated with the host shell and the lines in between are input or output unique to NodeBrain. The `define` command is highlighted, indicating it is the focus of the example. Lines ending with a backslash `\` indicate when a command is continued on the next displayed line. This is supported by the language within source files, but not for other methods of command input. If you copy an example of a command displayed over multiple lines, you must enter it as a single line when used outside the context of a source file.

```
$ nb
> define myFirstRule on(a=1 and b=2) mood="happy";
> assert mood="sad";
> show mood
mood = "sad"
> assert a=1,b=2,c=3,d="This is an example of a long single line that",\
    e="we depict on multiple lines to fit on the documnet page";
2008/06/05 12:09:08 NB000I Rule myFirstRule fired(mood="happy")
> show mood
mood = "happy"
> quit
$
```

# Table of Contents

# 1 Concepts

The Baseline module is a NodeBrain plug-in for statistical anomaly detection. It enables the use of nodes that maintain a simple statistical profile for a set of measures. A Baseline node monitors the current value of each measure and alerts when a measure is considered an anomaly relative to the statistical profile.

## 1.1 Average Value

An exponentially weighted moving average (EWMA) is maintained for each measure. This is a common technique for calculating an average with a different weight given to the more recent values. It also has the advantage of not requiring the storage of past values.

```
a[i] = w v[i] + (1-w) a[i-1]    where a is EWMA, w is a weight 0<=w<=1, and v is a value
```

The contribution of a new value to the average is determined by the weight `w`. The weight also determines the rate at which this contribution decays as new values are included. Larger values of `w` give more input to a new value, but also cause this contribution to decay more quickly as new values are included.

## 1.2 Deviation

The difference between a new value and the current average is called the deviation.

```
d[i] = v[i] - a[i-1]
```

You can restate the formula for the exponentially weighted moving average as an adjustment using the product of weight and deviation.

```
a[i] = a[i-1] + w d[i]
```

## 1.3 Average Deviation

The average deviation of actual values from the expected values is also computed as an EWMA. This is an approach for measuring the amount of variation in a measure. It is calculated like the average except the value is replaced by the absolute difference between the new value and prior average value.

```
D[i] = w abs(d[i]) + (1-w) D[i-1]     where D is the average deviation
```

Restating this as an adjustment, it looks like this.

```
D[i] = D[i-1] + w abs(d[i]-D[i-1])
```

## 1.4 Standard Deviation

The average deviation is expected to be 0.8 of the standard deviation for a normal distribution. Use this to approximate a standard deviation by multiplying the average deviation by 1.25. Specify the threshold as a factor times the approximate standard deviation. It is common to consider a value more than three standard deviations (3-sigma) an anomaly. For a normal distribution, 3-sigma will include 97 percent of the values in the normal range.

```
s = 1.25 D     where D is the average deviation and s is an approximate standard deviation
```

## 1.5 Weight

A Baseline node is assigned a weight factor from 0 to 1 used in the calculation of the average value and average deviation. Think of this as a level of skepticism about the latest value or deviation being the new "normal." The lower the value, the slower the Baseline node will adapt the expectation for any given measure based on new information. A value of 0 can be used to avoid "learning" from new information. This is appropriate in cases where a "normal" range of values is known and there is no reason to allow actual values to alter the definition of "normal."

On the other extreme, a value of 1 will cause the Baseline node to use the last value as the definition of "normal" when assessing a new value. Values from 0.2 to 0.3 are often used when "learning" is desired, but there is a reasonable level of skepticism about the latest value representing a new "normal."

## 1.6 Tolerance

A Baseline node is assigned a tolerance factor that specifies how "abnormal" a value must be to consider it an "anomaly" worthy of an alert. This factor, a single value for all measures, is expressed in units of standard deviation, a value specific to each measure. For example, a value of 3 is used to establish a threshold of 3 standard deviations (3-sigma), which establishes control limits in steps of 3 standard deviations above and below the average.

## 1.7 Thresholds

A threshold is an amount of deviation from the average, beyond which a new value is considered an anomaly. You can define a threshold unit for any given measure as the tolerance times the standard deviation.

```
u[i] = t s[i-1]       where u is the threshold unit, t is the tolerance, and s is the approx-
imate standard deviation
u[i] = 1.25 t D[i-1]  substituting 1.25 D for s
u[i] = T D[i-1]       where T is an internal tolerance factor 1.25 t
```

A threshold is established at every positive integer multiple of the threshold unit. The first threshold is at 1u, the second at 2u, and so on.

## 1.8 Anomaly Level

For each measure, a Baseline node maintains a current anomaly level, which is the last deviation divided by the threshold unit rounded down to the nearest integer.

```
L[i] = int( d[i-1] / u[i-1] )
```

Alerts are triggered when the anomaly level of a measure increases.

## 1.9 Limits

A control process has upper and lower limits that define the range of normal values. Values outside these limits are considered anomalies. Define the upper and lower limits in threshold units above and below the average. These limits identify the values where alerts are triggered, although this is simply a restatement of deviation thresholds.

To avoid triggering alerts for each new data point during an anomalous episode (period of abnormal values), the upper and lower limits open when the anomaly level increases (this is when an alert triggers) and close down when the anomaly level decreases. In the formulas below, `a` is the average value, `n` = 1 + `L`, where `L` is the anomaly level, and `u` is the threshold unit.

```
UL = a + n u     upper limit
LL = a - n u     lower limit
```

## 1.10 Cycle Duration

It is common for measures to have patterns of periodic variation. For example, it is normal for the temperature to be lower at night than during the day. It is normal for user access to a computer to increase during user working hours. To take this normal variation into account, a Baseline node is told the cycle time of the repeating pattern, expressed in minutes. If it is a daily cycle, then the duration is 24*60 minutes. If it is a weekly cycle, then the duration is 7*24*60 minutes.

## 1.11 Period Duration

The cycle time is divided into shorter periods of equal duration. The statistical profile maintained by a Baseline node includes an average value and average deviation for each measure for each period. If a one-hour period is specified (60 minutes) with a daily cycle, there will be 24 periods for which separate statistics are maintained. If the cycle is weekly, then a 60-minute period duration will generate 168 periods with separate statistics.

## 1.12 Statistical Profiles

A Baseline statistical profile is maintained as a set of files in a directory. Each file contains a profile for a single period, providing the average value and average deviation for each measure. A period profile is just a set of Baseline node commands.

```
.(arg1,arg2,...):set averageValue,averageDeviation;
...
```

Here's an example with two measures.

```
.("cpu utilization"):set 60.456,10.45;
.("disk utilization"):set 50.48,20.984;
```

The filename of a period profile is based on the number of seconds from the start of a cycle.

```
nnnnnnnn.nb
```

If a one-hour period is used, the filenames will increment by 60*60 or 3600. At any given UTC time, you can compute which profile to use as follows.

```
t =: UTC time
c =: cycle duration
p =: period duration
T = t % c              time within the cycle
F = int ( T / p ) * p   file time
```

# 2 Tutorial

*Do not put your faith in what statistics say until you have carefully considered what they do not say.* —William W. Watt

A Baseline node is used to detect statistical anomalies. It does not tell us what is good, bad, or important. However, it can help to find conditions worthy of further investigation.

## 2.1 Creating Profile Directory

Create a directory for storing period profiles. Since you are going to create a statistical profile for weather measurements, name the directory `weather`, and create it in the `cache/baseline` subdirectory of the caboodle (NodeBrain application directory).

```
$ cd CABOODLE
$ mkdir -p cache/baseline/weather
```

## 2.2 Constant Expectation

Let's say you are monitoring the temperature and humidity inside a house where it is expected to be relatively constant over time. There is no need to manage multiple period profiles, nor is there a reason to learn a statistical definition of normal. Instead, you can use a single period with a static profile that you define manually.

```
# cache/baseline/weather/00000000.nb
.("temperature"):set 68,2;
.("humidity"):set 45,5;
```

In this case, define the Baseline node as follows.

```
# arguments: directory, weight, tolerance, cycle, period
define indoor node baseline("cache/baseline/weather",0,0.8,60,60):static;
```

Here's a quick explanation of your choices.

- The `static` option is used to avoid updating the profile at the end of a period.
- A weight of 0 is used to avoid adjusting the average based on new actual values. This is not technically required under the `static` option, but any other value would be misleading.
- A tolerance of 0.8 is specified so you can express the range of normal values in the same units as the measure. This may seem a bit odd, but because of the approximation of standard deviation, 0.8 standard deviation is 1 average deviation. Since the profile contains average deviation, you can set it to half the normal range and set the average to the center of the normal range.
- A one-hour (60 minute) cycle and period are used so only one period profile is needed. Since the period duration divides into the cycle duration one time, keep repeating a single period.

## 2.3 Asserting Values

There is nothing special about the way to make assertions to a Baseline node. You assert values in the same way you would assert to a Tree node. Here are some examples.

```
indoor. assert ("temperature")=67.4,("humidity")=45.3;
assert x=1,y="abc",indoor("temperature")=67.4,indoor("humidity")=45.3;
indoor. assert x=2,("temperature")=73;
```

## 2.4 Anomaly Response Rules

Given the last assertion above, where temperature is asserted to be 73, you may expect the following alert to be generated by the "indoor" node shown previously.

```
alert _measure="'temperature'",_value=73,_average=68,_sigma=2.5,_deviation=5, \
      _threshold=5,_level=1;
```

By itself this alert does nothing. You need to provide rules to specify a response. Here's an example that simply echoes an entry out to an application log, and when it is a temperature anomaly also logs to the system log.

```
# arguments: directory, weight, tolerance, cycle, period
define indoor node baseline("cache/baseline/weather",0,0.8,60,60):static;
# alert _measure="...",_value=n,_average=n,_sigma=n,_deviation=n,_threshold=n,_level=n;
indoor. define logecho if(1):$ =echo `date` "${_measure} value of ${_value} is out of  \
                                range - threshold is ${_threshold}" >> weather.log
indoor. define logger if(_measure="'temperature'"):$ =logger -p local0.notice -t WEATHER \
                                 "${_measure}=${_value} threshold=${threshold}"
```

Here's another example where the response is more complicated. The `alarmit` rule invokes the `alarm` macro, which is defined on the third line. The macro makes an assertion to the `alarmMessage` cache. The cache logs a message to the system log, but only when the measure has had no anomaly alerts for at least 1 hour prior. The `cacheit` rule asserts the measure name to the `anomaly` cache, which can be used to check whether a measure has had an anomaly alert in the past 2 hours.

```
define alarmMessage node cache(~(h):measure,message(1));
alarmMessage. define alarmit if(message._hitState):$ =logger -p local0.notice -t WEATHER \
                                                                        "${message}"

define alarm macro(msgid,measure,text):% alarmMessage. assert ("%{measure}","%{msgid} %{text}");

# keep track of anomalies within the past 2 hours
define anomaly node cache:(~(2h):measure);

# arguments: directory, weight, tolerance, cycle, period
define indoor node baseline("cache/baseline/weather",0,0.8,60,60):static;
# alert _measure="...",_value=n,_average=n,_sigma=n,_deviation=n,_threshold=n,_level=n;
indoor. define alarmit if(1):$ $alarm("XYZ0001","${_measure}", \
                   "Weather measure ${_measure}=${_value} threshold=${_threshold}");
indoor. define cacheit if(1) anomaly(_measure);
```

## 2.5 Periodicity

Now let's suppose you live in a barn, where the temperature and humidity are not so constant, but where the temperature values are expected to vary by time of day. You may want a profile for every 4-hour period during the day. There would be six periods per day. The period profiles are stored in files with names incrementing by 4*60*60, or 14400 seconds.

| **Period** | **Profile** |
|---|---|
| 00:00 to 04:00 | 00000000.nb |
| 04:00 to 08:00 | 00014400.nb |
| 08:00 to 12:00 | 00028800.nb |
| 12:00 to 16:00 | 00043200.nb |
| 16:00 to 20:00 | 00057600.nb |
| 20:00 to 00:00 | 00072000.nb |

The Baseline node now has a 24-hour cycle time and a 4-hour period.

```
# arguments: directory, weight, tolerance, cycle, period
define indoor node baseline("cache/baseline/weather",0,0.8,24*60,4*60):static;
```

## 2.6 Learning

Suppose you don't have a preconceived notion of what a normal range of values is for your measures. In this case, you want to let the Baseline module create the profiles for you and adjust the average value and average deviation based on actual values experience while learning is enabled. To do this, simply remove the `static` option from the Baseline node definition and provide a non-zero weight to control how fast to adapt averages to new actuals. Because you no longer know the range of normal values, you express the tolerance in standard deviations (e.g., 3 below).

```
# arguments: directory, weight, tolerance, cycle, period
define indoor node baseline("cache/baseline/weather",0.2,3,24*60,4*60);
```

At the end of each period, the profile is updated with weight-adjusted averages for each measure's value and deviation.

## 2.7 Computer Network Periods

Measurements within a computer network (e.g., number of packets by protocol) typically vary based on user and software schedules that vary by time of day and day of week. If you are monitoring measures with this type of periodicity, it is appropriate to use a weekly cycle and hourly period.

```
# arguments: directory, weight, tolerance, cycle, period
define packets node baseline("cache/baseline/packetsbyprotocol",0.2,3,7*24*60,60);
```

## 2.8 Summation

When your measures are counts (e.g., number of packets by protocol), your information source may provide counts over units of time much smaller than the period duration. In this case, you want to sum up the counts from the information source over the period duration. You do this by specifying the `sum` option.

```
# arguments: directory, weight, tolerance, cycle, period
define packets node baseline("cache/baseline/packetsbyprotocol",0.2,3,7*24*60,60):sum;
```

Now when you assert values to measures within the Baseline node, the measures are incremented instead of being set to the new value.

```
packets. assert ("icmp")=465,("http")=1024,("smtp")=34; # increment measures
```

At the end of each period, the measures are all reset to zero when the `sum` option is used.

# 3 Commands

This section describes the syntax and semantics of commands used with the Baseline module.

## 3.1 Define

This section describes define commands the Baseline module supports.

### 3.1.1 Define Baseline

The `define` command is used to create a Baseline node.

```
define node node baseline("directory",weight,tolerance,cycle,period)[:options];
```

| **Parameters** | **Description** |
|---|---|
| *directory* | Path of directory for storing baseline files called period profiles. This should normally be a relative path within a caboodle, and the "cache" directory (e.g., "cache/baseline/*baseline*") is recommended. You should avoid having more than one node, in one or more agents, referencing the same Baseline profile directory. |
| *weight* | A real number from 0 to 1 that determines how much weight should be given to a new value relative to past values. |
| *tolerance* | The number of standard deviations from the average for which a given deviation is considered to be within a normal range. |
| *cycle* | Number of minutes in each cycle for which a pattern of periodic variation is expected. Examples are daily (24*60) and weekly (7*24*60). |
| *period* | Number of minutes in each period within a cycle for which an average value and average deviation is maintained for each measure. |

| **Option** | **Description** |
|---|---|
| found | The `found` option is assigned a cell expression to be used as a default value when an element is found by an evaluation but has no value. This happens when an evaluation has fewer arguments than the assertion that created the element. For example, if you assert `foobar("abc","def")=5` and evaluate `foobar("abc")`, the "abc" element is found, but has no value, assuming it was not explicitly asserted also. By default, ? (unknown) is returned. |

```
found=cellExpression
```

| | |
|---|---|
| notfound | The `notfound` option provides a default value when an element is not found during an evaluation. The default value for `notfound` is ? (unknown). A value of 0 can be used to implement the "closed world assumption," where anything not known to be true is assumed to be false. |

```
notfound=cellExpression
```

| | |
|---|---|
| order | The `order` option specifies the use of argument values for comparison when searching and maintaining binary trees within a Baseline node. By default, comparisons are performed on the address of argument value objects for added efficiency. This is possible because NodeBrain maintains only one object for any given value. |
| partition | The `partition` option implies the `order` option and changes the evaluation operation to match an argument to the element with the maximum value less than or equal to the argument value. This enables a domain of values to be partitioned into ranges of values. For example, a partitioned Baseline node might be used to monitor the number of traffic accidents by driver age ranges in 10-year increments. |
| static | Use `static` to avoid updating the profile at the end of each period. This is appropriate when the profile defines the acceptable range of values for each measure, and "learning" is not desired. Although the weight can be set to 0 to avoid adjusting average value and average deviation statistics in a profile for known elements, this does not prevent the Baseline node from updating the profile with new elements. Use of the `static` option prevents the addition of new elements, and (if weight is not zero) enables learning for elements not included in the profile. However, learning for an element that is not in the profile develops statistics over all periods, instead of per period. |
| sum | When `sum` is used, asserted values are summed up over each period. The period value is then the sum of all values asserted during the period. When the sum reaches an upper limit and the assertion and average are both positive, or a lower limit and the assertion and average are both negative, an alert can be triggered right away. However, a lower limit for positive average and upper limit for negative average must be checked at the end of a period when the full sum is known. |
| trace | The `trace` option is used to generate log messages for troubleshooting. |

## 3.2 Assertions

The Baseline module supports assertions within `assert` or `alert` commands.

```
assert node(arg1[,arg2,...])[=value];  # value must be number or ? (Unknown)
```

The semantics are similar to the Tree module, except the value assigned to an element within a Baseline node must be a number. Strings are not allowed. When the `sum` option is used by a Baseline node, an assertion to a Baseline node will add to the current value, otherwise it replaces the current value. If no value is specified, 1 is assumed. If an unknown value ("?") is assigned, the element is removed.

## 3.3 Cell Expressions

When a Baseline node is referenced in a cell expression, it presents the value of the specified element. The example shows a reference within a cell expression of a simple `define` statement.

```
define term cell node(arg1[,arg2,...]);  # term has value of specified element
```

## 3.4 Node Commands

This section describes commands implemented by the Baseline module for use with defined nodes.

```
node[(arg1[,arg2,...])]:verb [arguments]
```

### 3.4.1 Balance

The `balance` command is used to rebalance the trees within a Baseline node after `flatten` or `prune` commands have made the trees unbalanced.

```
node:balance
```

### 3.4.2 Flatten

The `balance` command is used to completely unbalance the trees within a Baseline node so they effectively become lists.

```
node:flatten
```

### 3.4.3 Prune

The `prune` command is used to remove elements from a Baseline node.

```
node[(arg1[,arg2,...])]:prune
```

Unlike an assertion that an element is unknown, which removes the identified element and the subordinate tree, the `prune` command only removes the subordinate tree. Using a table model, it deletes the subordinate table, emptying the cells to the right of the identified cell and deleting all rows represented by the subordinate table.

### 3.4.4 Set

The `set` command is used to assign the average value and average deviation for an element in a Baseline node.

```
node(arg1[,arg2,...]):set averageValue,averageDeviation;
```

Although `set` commands can be issued at any time, they are primarily used in period profiles. A period profile is just a file containing `set` commands. These files are processed ("sourced") at the start of a period and optionally updated at the end of a period.

### 3.4.5 Store

The `store` command is used to write the current measures within a Baseline node to a file in the form of assertions.

```
assert (arg1[,arg2,...])=value;
...
```

The node name is not included in the assertions. This enables the assertions to be easily applied to a different node—perhaps not even a Baseline node.

### 3.4.6 Trace

The `trace` command is used to toggle the `trace` option for troubleshooting.

```
node:trace [on|off]
```

## 3.5 Module Commands

The Baseline module currently implements no module level commands.

# 4 Triggers

When measures reach thresholds, a Baseline node issues an alert to `if` rules defined for the node. These alerts have the following form.

```
alert _measure="name",_value=n,_average=n,_deviation=n,_threshold=n,_limit=n, \
      _sigma=n,_level=n;
```

| **Attribute** | **Description** |
|---|---|
| _measure | Name of the measure for which the value is outside the specified limit. |
| _value | Value of the measure outside the limit. |
| _average | Learned average value for the period. |
| _deviation | Absolute value of the different between actual (_value) and expected (_average). |
| _threshold | Deviation threshold surpassed. |
| _limit | Control limit surpassed. This is the average plus or minus the threshold. |
| _sigma | Approximate standard deviation for the period. |
| _level | Magnitude of anomaly. threshold=(2**level)*tolerance*sigma |

# Index