# Servant NodeBrain Module

**Release 0.9.02**

Servant NodeBrain Module
August 2014
NodeBrain Open Source Project

**Release 0.9.02**

Author: Ed Trettevik



**MIT License**



**NodeBrain License**

**History**

2005-10-12 Title: *Servant NodeBrain Module*
Author: Ed Trettevik <eat@nodebrain.org>
Publisher: NodeBrain Open Source Project

2010-12-31 Release 0.8.10

- Updates - still needed

## Preface

This tutorial is intended for readers seeking an introduction to NodeBrain through a series of simple examples. Other documents are available for readers looking for a more complete reference to the rule language, modules, or API (application programmatic interface).

The intent of the examples in this tutorial is to illustrate individual concepts, not to provide complete working applications or show all related options. We avoid formal syntax descriptions, thinking you are here because you want to figure it out from examples.

Files referenced in this tutorial are included in the tutorial directory of the NodeBrain distribution.

See www.nodebrain.org for more information and the latest update to this document.

## 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 ..... 1**

1.1 Servant Command ..... 1
1.2 Servant Node ..... 1
1.3 NodeBrain as Servant ..... 1

**2 Tutorial ..... 3**

2.1 Creating a Servant Program ..... 3
2.2 Specifying a Servant Node ..... 3
2.3 Execution ..... 3

**3 Commands ..... 5**

3.1 Define ..... 5
3.2 Node Commands ..... 5

**4 Triggers ..... 7**

**Index ..... 9**

# 1 Concepts

In NodeBrain terminology, a servant is a shell command executed as a child process to NodeBrain. A servant may communicate with NodeBrain via the servant's `stdin` and `stdout`. Servants makes it easy for programmers to develop in their favorite programming languages when interfacing to NodeBrain rules. Scripting languages like Perl are convenient for creating servants. You should only consider writing new NodeBrain modules when a problem cannot be solved as well by writing a servant.

## 1.1 Servant Command

A servant command (`"-"` or `"="`) is a convenient may to execute a shell command as a child process. The command may perform a function that does not require communication with NodeBrain, or it may return commands and log messages for NodeBrain to process. See the *NodeBrain Language Reference* for the syntax of servant commands (`"-"` and `"="`).

## 1.2 Servant Node

A servant node, implemented by the Servant module, is also used to execute a shell command as a child process. But a servant node is different than a servant command in the following ways.

1. Rules can send messages to a servant node on the servant's `stdin` in addition to accepting commands on the servant's `stdout`.
2. Servant nodes are not enabled (spawned) until NodeBrain enters a server mode. This is necessary in the case of daemons (-d) because the NodeBrain process terminates when it spawns itself as a daemon. Any processes started before entering daemon mode are orphaned and no longer able to communicate on `stdin` or `stdout`. This is also helpful for other server modes (e.g., servant) because it enables all the startup rules to be fully defined before a servant starts generating commands.
3. In a future release you will be able to use the `enable` and `disable` commands to start and stop servant nodes. This is possible because they have a term (name) that can be used as a reference. The servant command (`"-"` or `"="`) only identifies a servant by process ID (PID), and although the PID is displayed and can be referenced in shell commands like `kill`, the PID is not captured in a way that NodeBrain rules can reference.

## 1.3 NodeBrain as Servant

A servant node can run NodeBrain (nb) in a new process the same way it runs other servant programs and scripts. The servant module's support for two-way message communication between processes enables a NodeBrain application to be divided into multiple processes. When you define a servant that executes NodeBrain, messages sent to `stdin` and received from `stdout` are all NodeBrain commands. This allows the processes to share information and make decisions or take actions as a service to each other.

When NodeBrain is run as a servant to another NodeBrain process, use the -s (servant) option if you want the child to run as a server. Use the `"="` option instead if you want the child to operate in batch mode.

The *NodeBrain Language Reference* provides a description of NodeBrain options and the full syntax of the servant command ("-" or "="), which is key to understanding the syntax used by the servant module.

# 2 Tutorial

> *In order to become the master, the politician poses as the servant.* —Charles de Gaulle (1890–1970)

NodeBrain is not intended to be the master of all things. As in politics, it is often more convenient to let a servant be the master. In this tutorial, you will learn how to create a servant in your favorite programming language to obtain information needed to make decisions. You will see that when a servant sends commands to NodeBrain, it becomes the master—like a politician once elected.

## 2.1 Creating a Servant Program

To keep it simple and only hint at something useful, let's create a servant script using Perl that pretends to tell you the cost of gas and bread, something every politician should be prepared to include in a campaign speech.

```
#!/usr/bin/perl
# File: tutorial/Servant/charles.pl
my $gas=2.50;
my $bread=1.10;
$|=1;
while(<>){
  chomp($_);
  if(/gas/){print("assert gas=$gas;\n");$gas+=.50;}
  elsif(/bread/){print("assert bread=$bread;\n");$bread+=.25}
  else{print("alert msg=\"item '$_' not recognized\";\n");}
  }
```

## 2.2 Specifying a Servant Node

Now you need some rules to use the servant program. Create a script that looks like this.

```
#!/usr/local/bin/nb -s
# File: tutorial/Servant/charles.nb
define price node servant:|=|:./charles.pl
define ouch on(gas>4 or bread>3):stop;
define getgasprice on(~(3s)):price:gas
define getbreadprice on(~(3s)):price:bread
```

The Servant node module specification includes an `=` command to specify the program and what to do with `stdout` and `stderr`. It also supports a leading pipe (`|`) to enable the sending of text to the program on `stdin`.

## 2.3 Execution

This script is designed to run like an agent without detaching from the terminal. The `-s` option is the trick. The script will pause for 3 seconds between scheduled events, so just be patient and the script will end when the price of one of the items gets too painful.

```
$ ./charles.nb
2008/08/21 19:08:28 NB000I Argument [2] ./charles.nb
> #!/usr/local/bin/nb -s
> # File: tutorial/Servant/charles.nb
> define price node servant:|=|:./charles.pl
> define ouch on(gas>4 or bread>3):stop;
> define getgasprice on(~(3s)):price:gas
> define getbreadprice on(~(3s)):price:bread
2008/08/21 19:08:28 NB000I Source file "./charles.nb" included. size=194
2008/08/21 19:08:28 NB000T Servant mode selected
---------- ----------------------------------------------------
2008/08/21 19:08:28 NM000I servant price: Enabling|=|:./charles.pl
2008/08/21 19:08:28 NM000I servant price: Enabled[21633] |=|:./charles.pl
2008/08/21 19:08:31 NB000I Rule getbreadprice fired
: price:bread
2008/08/21 19:08:31 NB000I Rule getgasprice fired
: price:gas
> price. assert bread=2.1;
> price. assert gas=3.25;
2008/08/21 19:08:34 NB000I Rule getbreadprice fired
: price:bread
2008/08/21 19:08:34 NB000I Rule getgasprice fired
: price:gas
> price. assert bread=2.35;
> price. assert gas=3.75;
2008/08/21 19:08:37 NB000I Rule getbreadprice fired
: price:bread
2008/08/21 19:08:37 NB000I Rule getgasprice fired
: price:gas
> price. assert bread=2.6;
> price. assert gas=4.25;
2008/08/21 19:08:37 NB000I Rule ouch fired
: stop;
2008/08/21 19:08:37 NB000I [21633] Killed(1)
2008/08/21 19:08:37 NB000I NodeBrain nb[21632] terminating - exit code=0
```

# 3 Commands

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

## 3.1 Define

A servant is defined as a node, where the foreign text includes the full syntax of a servant command ("-" or "=").

**Syntax**

| | |
|---|---|
| *servantDefinition* | ::= **define** š *term* š **node** [š *servantSpec* ] • |
| *servantSpec* | ::= **servant** [ **(** *servantOptions* **(** ] **:** [ **|** ] *servantCmd* |
| *servantOptions* | ::= there are currently no options defined for this module |
| *servantCmd* | ::= see "-" and "=" command in the *NodeBrain Language Reference* |

## 3.2 Node Commands

Once NodeBrain is running in a server mode (e.g., agent/daemon/service or servant), you may pass messages to a servant using a node command. Suppose you define a Perl script named `servant.pl` as a servant named *servant*.

```
define servant node servant:|=:./servant.pl
```

Once in server mode, you can pass a message to `servant.pl` on `stdin` as follows.

```
servant:message
```

This is how you pass messages to any node. In this case, the servant node module simply forwards all messages to the specified servant program on `stdin`.

# 4 Triggers

There are no triggers provided by the Servant module directly. It is entirely up to the servant program to determine what commands are sent to NodeBrain and the conditions that trigger them. Output from a servant may be interpreted as NodeBrain commands in the servant's context. This is no different than receiving messages from a servant command ("-:" or "=:"). In both cases, the messages are interpreted as NodeBrain commands within the context in which they are specified.

Let's look at a trivial example. Suppose the `servant.pl` looks like this.

```
#!/usr/bin/perl
use FileHandle;
STDOUT->autoflush(1);  # flush stdout as soon as we write to it
while(<>){
   if(/^on$/){print("assert status;\n");}
   elsif(/^off$/){print(assert !status;\n);}
   else{print("assert ?status;\n");}
   }
```

Now let's define the following NodeBrain daemon script and call it `servant.nb`.

```
#!/usr/local/bin/nb -d
set log="servant.log";
-cp /dev/null servant.txt
define taylor   node servant:=:tail -f servant.txt
define servant node servant:|=:./servant.pl
servant. define hello on(.status):-echo "hi"
```

The following commands will cause the daemon to echo "hi" to the log file twice.

```
$ ./servant.nb
$ echo "servant:on" >> servant.txt
$ echo "servant:off" >> servant.txt
$ echo "servant:on" >> servant.txt
```

The servant named "taylor" uses the `tail` command to input anything you write to `servant.txt`. You emptied it out in the line before the servant definition to prevent `tail` from giving you old commands from a previous session. When you echo the NodeBrain command (e.g., "servant:on") to `servant.txt`, `tail` sends them to the servant node module, which passes them on to the NodeBrain interpreter. These commands happen to address the second servant, the one executing the `servant.pl` script. The values of "on," "off," and "on" are sent to this script on `stdin`. The `servant.pl` script simply translates these messages into NodeBrain commands that set a value for the "status" term. When status is set to 1, the "hello" rule fires.

Obviously the `servant.pl` script in this example isn't giving you any information you don't already know, so it is of little or no value. You could have just echoed "servant.status=1;" to `servant.txt` in place of "servant:on." But the goal here is just to illustrate the servant

interface. It should not be difficult to imagine more realistic situations where you send messages to a servant instructing it when to provide information or what information to provide. For example, the servant could check to see whether a process is running or how much space is available in a file system. Servants just need to know how to perform a function or get some information. The NodeBrain rules decide how to respond to the information provided and when specific actions are appropriate.

# Index


## C

commands . . . . . 5
concepts . . . . . 1

## D

define command . . . . . 5

## N

node commands . . . . . 5
NodeBrain as Servant . . . . . 1

## S

servant command . . . . . 1
servant node . . . . . 1

## T

triggers . . . . . 7
tutorial . . . . . 3