Pipe NodeBrain Module

Release 0.9.02

Pipe NodeBrain Module
August 2014
NodeBrain Open Source Project

Release 0.9.02

Author: Ed Trettevik

Copyright © 2014 Ed Trettevik <eat@nodebrain.org>

Permission is granted to copy, distribute and/or modify this document under the terms of either the MIT License (Expat) or the NodeBrain License.

MIT License

Copyright © 2014 Ed Trettevik <eat@nodebrain.org>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

NodeBrain License

Copyright © 2014 Ed Trettevik <eat@nodebrain.org>

Permission to use and redistribute with or without fee, in source and binary forms, with or without modification, is granted free of charge to any person obtaining a copy of this software and included documentation, provided that the above copyright notice, this permission notice, and the following disclaimer are retained with source files and reproduced in documention included with source and binary distributions.

Unless required by applicable law or agreed to in writing, this software is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

History

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

2010-12-31 Release 0.8.3

- 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

The Pipe module provides a simple unauthenticated method of communication between NodeBrain and external programs running on the same system. A pipe server node is used to accept input from other programs while a pipe node is used to write output to another program. You must use file permissions on the pipe (FIFO) file for security. An identity is associated with input commands to enable restrictions on the input commands.

2 Tutorial

> *To listen closely and reply well is the highest perfection we are able to attain in the art of conversation.* —Francois de La Rochefoucauld (1613–1680)

To construct a successful NodeBrain application, you must configure NodeBrain to listen closely and respond well. The *Operating Mode* tutorial illustrates how NodeBrain can listen for input commands from `stdin` and both input commands and error messages from servant scripts. NodeBrain's ability to listen is controlled by a component called the *medulla.* Node modules interface with the medulla to extend NodeBrain's ability to listen to include other sources of input.

A pipe node is perhaps the simplest of listening nodes. It listens to a FIFO (named pipe) file, and you can write to the pipe using any program you like, including an `echo` command.

2.1 Defining Pipe Server Nodes

To begin this tutorial, create an agent script called `smokey.nb` with two pipe server nodes as shown here.

```
#!/usr/local/bin/nb -d
# File: tutorial/Pipe/smokey.nb
-rm smokey.log
setlog="smokey.log",out=".";
declare jed identity guest;
declare chief identity owner;
define corncob node pipe.server("jed@corncob");
corncob.define r1 on(a=1 and b=2);
define peace node pipe.server("chief@peace");
peace.define r1 on(a=1 and b=2);
```

The argument to a `pipe.server` is of the form "*identity*`@`*pipe*". Identities are associated with listening nodes to limit the types of commands the interpreter will accept from the node. In this example, you associate the identity `jed` with the corncob pipe, and the identity `chief` with the peace pipe. You have declared `jed` to be a guest and `chief` to be an owner. A guest can only connect and issue `show` commands—like read only access. An owner can issue any command, including shell commands, which means they have all permissions of the user that started the agent. For this reason, pipes are created with owner-only read/write permissions. However, if you declare the associated identity to be a "peer," you can give other user's write permission on the pipe. They will be able to issue assertions and alerts but not modify your rules or issue shell commands. You must still think through how the agent will respond to their assertions. For example, if you create a rule that reboots the system when `a=1`, then letting someone assert `a=1` is the same as letting them reboot the system.

2.2 Starting Pipe Server Agent

Now let's start the agent. This script is executable because the `-d` (daemon) option is specificied on the she-bang line. So you can just execute it like any executable and it will load the rules and go into the background (daemonize).

```
$ ./smokey.nb
2008/06/1017:09:16 NB000I Argument [1] -d
2008/06/1017:09:16 NB000I Argument [2] ./smokey.nb
> #!/usr/local/bin/nb -d
> # File: smokey.nb
> -rm smokey.log
[13993]Started: -rm smokey.log
[13993]Exit(0)
> set log="smokey.log",out=".";
2008/06/1017:09:16 NB000I NodeBrain nb will log to smokey.log
> declare jed identity guest;
> declare chief identity owner;
> define corncob node pipe.server("jed@corncob");
> corncob. define r1 on(a=1 and b=2);
> define peace node pipe.server("chief@peace");
> peace. define r1 on(a=1 and b=2);
2008/06/1017:09:16 NB000I Source file "./smokey.nb" included. size=323
2008/06/1017:09:16 NB000I NodeBrain nb[13992,4118] daemonizing
$
```

2.3 Sending Commands to Pipe Server Nodes

Use the `echo` command to send NodeBrain commands to the pipe servers.

```
$ echo stop > corncob
$ echo stop > peace
```

The log file shows you what happened.

```
$ cat smokey.log

N o d e B r a i n   0.9.02 (Columbo) 2014-02-15

Compiled Jun 12 2014 19:20:12 x86_64-unknown-linux-gnu

Copyright (C) 2014 Ed Trettevik <eat@nodebrain.org>
MIT or NodeBrain License
----------------------------------------------------------------

/usr/local/bin/nb -d ./smokey.nb

Date       Time     Message
---------- -------- ------------------------------------------------
2014-06-10 17:09:16 NB000I NodeBrain nb[13994:1] myuser@myhost
2014-06-10 17:09:16 NB000I Agent log is smokey.log
2014-06-10 17:09:16 NM000I pipe.server peace: Listening for FIFO connections as chief@peace
2014-06-10 17:09:16 NM000I pipe.server corncob: ...
... Listening for FIFO connections as jed@corncob
2014-06-10 17:41:11 NM000I pipe.server corncob: FIFO jed@corncob
> corncob. stop
2014-06-10 17:41:11 NB000E Identity "jed" does not have authority to issue stop command.
2014-06-10 17:41:19 NM000I pipe.server peace: FIFO chief@peace
> peace. stop
2014-06-10 17:41:19 NB000I NodeBrain nb[13994] terminating - exit code=0
$
```

Notice that when you sent a `stop` command to the corncob pipe, you didn't have the needed authority, but when you sent the same `stop` command to the peace pipe it worked.

Run `smokey.nb` again with the following commands to see what happens.

```
$ ./smokey.nb
$ echo "common. define r1 on(a=1 and b=2);" > peace
$ echo "common. assert a=1,b=3;" > peace
$ echo "common. assert b=2;" > peace
$ echo "common. show -t" > peace
$ echo "stop" > peace
$ cat smokey.log
```

3 Commands

This sections describes commands used with the Pipe module.

3.1 Define

Syntax

pipeServerDefineCmd
::= **define** š *term* š **node** [š pipeServerDef] •

pipeServerDef
::= **pipe.server("***identity***@***fileName***");**

Identify ::= name of identity to associate with input commands

Filename ::= name of input pipe (FIFO)

The identity must be declared prior to pipe server definition. The FIFO identified by *fileName* must exist when a pipe server is enabled.

```
declare fred identity ... ;
define mypipe node pipe(fred@pipe/myhose);
... include rules here...
```

3.1.1 Pipe Server Commands

The ENABLE and DISABLE commands are supported. Node commands and assertions are not currently supported.

3.1.2 Pipe Server Rules

Commands read from the pipe file are interpreted in the context of the pipe server node. Rules defined in this context must be designed for compatibility with the commands written to the pipe by the external program.

3.1.3 Pipe Definition

Syntax

pipeDefineCmd
::= **define** š *term* š **node** [š *pipeDef*] •

pipeDef ::= **pipe(** *fileName* **);**

Filename ::= name of output pipe (FIFO)

The FIFO identified by *fileName* must exist when a command is issued to the pipe node.

```
define mypipe node pipe("pipe/myhose");
```

3.1.4 Pipe Commands

Commands directed to a pipe node are simply written to the pipe (FIFO) file.

```
node:... any text...
```

The `ENABLE` and `DISABLE` commands and assertions are not supported.

4 Triggers

This module does not implement triggers. Each line received on a pipe is passed to the interpretter.

Index