System NodeBrain Kit

Release 0.8.17

System NodeBrain Kit
July 2014
NodeBrain Open Source Project

Release 0.8.17

Author: Ed Trettevik

MIT License

NodeBrain License

History

2007-11-05 Title: *System NodeBrain Kit*
Author: Ed Trettevik <eat@nodebrain.org>
Publisher: NodeBrain Open Source Project

Version 0.6.8 (not released)

- This document, like the software it describes, is a prototype.

2013-02-05 Release 0.8.13

- Converted to Texinfo
- Several updates in preparation for release

2013-05-06 Release 0.8.14

- Updates to *Agents*, *Adapters* and *Servants* chapters.

Preface

This document describes the System NodeBrain Kit. It is intended for users and developers of NodeBrain application kits. The reader should be familiar with basic NodeBrain concepts as covered in the *NodeBrain Tutorial*.

The System NodeBrain Kit derived from a tool developed in 1998 called the Unix System Monitor Kit, or Sysmon. The framework components of the original tool have evolved into the Caboodle NodeBrain Kit, while the application components have evolved into the System NodeBrain Kit. The original Unix System Monitor Kit (Sysmon) supported options for monitoring without a NodeBrain rule engine, substituting cron or a system management agent like HP Operations Manager to schedule the execution of probes. These options have been removed from the System NodeBrain Kit, requiring a NodeBrain rule engine, although integration with commercial system management agents is still possible and often desirable.

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

Documents

NodeBrain Tutorial
NodeBrain Language Reference

Document Conventions

Sample code and input/output examples are displayed in a monospace font and enclosed in a box. 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

This chapter introduces basic concepts of the System NodeBrain Kit. The reader should be familiar with the concepts introduced by the Caboodle NodeBrain Kit, upon which the System Kit depends.

1.1 System Agent

The System NodeBrain Kit provides a NodeBrain agent called the System agent, that monitors various elements of a server or application and issues alarms to the Caboodle agent provided by the Caboodle NodeBrain Kit. A server may run any number of System agents, each dedicated to monitoring the server, a shared application, or an application managed by a single user.

A small minimal set of monitoring plans is provided by the System Kit. Users are encouraged to develop additional kits that build on the System Kit and make them available to others. If you develop general monitors that seem like logical additions to the System NodeBrain Kit, let us know and your recommendation will be considered.

1.2 Server Monitoring

This kit can be used to monitor a server while running as `root` or as a dedicated user. A startup/shutdown script called "sysmon" is provided for this use case on Linux servers. The NodeBrain Open Source Project does not provide platform specific kits for monitoring servers, instead only a skeletal kit is provided. Users of the System Kit are encouraged to develop additional kits built on top of the System Kit, and to share those kits with others. For example, a robust set of log monitoring plans for a given platform would make a very useful kit.

For security reasons, when running NodeBrain as `root` or any account with special systemr-wide permissions, you should avoid using node modules that enable remote control.

1.3 Application Monitoring

Applications often have unique monitoring requirements that are not satisifed by server level monitoring. The System Kit may be used to implement application sepecific monitoring. When developing monitoring plans for generic applications used by many, consider reusability so others can benefit from your work.

2 Installation

The System Kit is released as a GNU style source distribution file, and an architecture independent RPM file, both of which can be downloaded at http://nodebrain.org, with SourceForge.net providing the download service.

2.1 GNU Source File

When installing from a GNU source distribution release file, follow these steps, adjusting the release number as needed to match the file you download.

```
# tar -xf nbkit-system-0.8.14.tar.gz
# cd nbkit-system-0.8.14
# ./configure
# make
# make check
# make install
```

When using this method, the kit is installed to '`/usr/local/share/nbkit/system-0.8.14.tar.gz`'. You can override this using `./configure --prefix=/home/foo`, which would cause the kit to be installed as '`/home/foo/share/nbkit/system-0.8.14.tar.gz`'.

If on a Linux platform that supports RPM files, you may issue the following command to create a source RPM file and a noarch RPM file to enable native package management.

```
# make rpm
```

2.2 RPM File

When installing from an RPM file, use the following rpm command. The RPM is not architecture dependent because it contains no compiled code, only Perl and NodeBrain scripts, and a few web pages and associated image files.

```
# rpm --install nbkit-system-0.8.14-1.noarch.rpm
```

When using this method, the kit is installed to '`/usr/share/nbkit/system-0.8.14.tar.gz`'.

3 Setup

This chapter walks the reader through the initial setup process as performed on a development system. The `nbkit` commands used in this chapter are explained in the Caboodle NodeBrain Kit manual and `nbkit` man page.

3.1 Adding System Kit to a Caboodle

To use the System Kit, you must already have a caboodle that uses the Caboodle NodeBrain Kit. To add the System Kit to your caboodle, issue the `use` command of `nbkit` as shown below. The `nbkit -k` command will display the kits installed on the development server. Specify the kit you want to use for the caboodle. The last line of the this example would be used for System Kit release 0.8.14 installed using an RPM file. However, the kit could just as easily come from a location within your home directory.

```
$ nbkit -k
$ nbkit caboodle use kit-directory
$ nbkit caboodle use /usr/share/nbkit/system-0.8.14.tar.gz
```

3.2 Enable System Folder

Plans provided by the System Kit are placed in the SystemFolder. To provide visibility of the SystemFolder when using the web interface, include it in the _Admin folder as follows.

```
$ nbkit caboodle enable SystemFolder._Admin.parent
```

3.3 Create System Agent Identity

The System agent runs with a NodeBrain identity, which must be established before starting the agent. Issue these commands.

```
$ nb
> peer.identify System
> quit
$ echo "declare System identity;" >> ~/.nb/user.nb
```

3.4 Start System Agent

Use the following command to start the System agent. This agent is responsible for monitoring the application provided by the caboodle.

```
$ nbkit caboodle start System
```

3.5 Plan Setup

To display plans provided by the System Kit, available for setup, issue the following `nbkit` command.

```
$ nbkit caboodle setup
```

Plans starting with "System" that are not already setup, may be setup as follows, where SystemProcess is an example.

```
$ nbkit caboodle setup plan
$ nbkit caboodle setup SystemProcess
```

3.6 Enabling Plans

System plans are designed as rules to be processed by the System agent. However, the System agent will only process them when they are enabled as children of the System agent.

```
$ nbkit caboodle enable plan.System.parent
$ nbkit caboodle enable SystemProcess.System.parent
```

To see the current relationships for a plan, use the `show` command.

```
$ nbkit caboodle show plan
$ nbkit caboodle show System
$ nbkit caboodle show SystemProcess
```

3.7 System Services

If your caboodle is intended as a system service, you can use a couple files provided by the System Kit in the setup directory to help on a Linux server. Here the steps are descibed as if you are performing the setup on a development server with a caboodle that uses the System Kit. Later you will automate these steps as part of your application package, perhaps an RPM file.

Let's say you are calling your service "foobar", and the caboodle is '`/var/foobar`'. Edit '`/var/foobar/setup/nodebrain.sysconfig`' and change `CABOODLE` to '`/var/foobar`' as illustrated below.

```
# Distributed as setup/nodebrain.sysconfig
CABOODLE=/var/foobar
AGENT=System
```

The value for `AGENT` will remain `System` if you are using the System agent within your caboodle to provide the foobar service. If the foobar service is provided by a different agent within our caboodle, then modify `AGENT` as well.

Perform the following commands to complete the setup.

```
# cp /var/foobar/setup/nodebrain.service /etc/init.d/foobar
# cp /var/foobar/setup/nodebrain.sysconfig /etc/sysconfig/foobar
# chkconfig add foobar
# service start foobar
```

Before converting a NodeBrain agent into a system service, issue a `stop` command using `nbkit`, even if the agent is not running. This establishes an intent for the service to be down with respect to the `nbkit` command.

```
$ nbkit foobar stop System
```

After converting a NodeBrain agent into a system service, you should perform all start, stop, and restart operations using the `service` command as `root`, and avoid using the `start`, `stop`, `check` and `bounce` options of the `nbkit` command on the agent.

```
# service foobar start
# service foobar stop
```

The problem with using options of `nbkit` to start or stop the agent is distruption of the `service` command's use of a pid file to identify a running service. (This is an opportunity for a future release of `nbkit` to include awareness of agents running as services, and to use the `service` command when appropriate.)

You can still use other `nbkit` commands like `edit` and `compile` on the agent. When the agent is compiled, either directly or as a result of a recursive compile started by a child plan change, restart the service.

```
# nbkit foobar compile System
# service foobar restart
```

4 Commands

The System Kit does not provide any commands. The commands provided by the Caboodle Kit are sufficient.

5 Agents

The System Kit provides only one agent called "System". This agent, in addition to the Caboodle agent provided by the Caboodle Kit, is often sufficient to monitor a server or application. However, you are free to create additional agents like the System agent to distribute monitoring tasks for any reason. Performance and availability are two reasons one might want to isolate one set of monitoring rules from another by using multiple agents.

6 Adapters

Adapers are small scripts that adapt NodeBrain to an application environment based on a model of interaction not defined by NodeBrain, but taking advantage of more general types of interaction defined by NodeBrain. In other words, adapters can fit into models defined by NodeBrain Kits, about which NodeBrain has no awareness.

6.1 Plan Compilers

The System Kit provides the following plan compilers to implement additional rule generation schemes. These schemes are described a bit under *Model Plans* later in this document. However, be best we to understand them is to study plans that use these schemes.

```
SystemAudit.compiler
SystemThreshold.compiler
SystemThresholds.compiler
```

6.2 Alarm Adapters

The System Kit does not provide any alarm adapters.

7 Servants

Servants connect NodeBrain rules to an application environment using a method of interaction prescribed by, and fully support by, NodeBrain. This sets them apart from adapters, which interact with NodeBrain indirectly, based on a model unknown to NodeBrain.

The System Kit provides a small set of servants, intended primarily to demonstrate a concept you can extend, but also to provide an application you can use without too much effort.

```
SystemCpu.probes
SystemFilesys.probe
SystemProcess.probe
SystemThreshold.node
```

These servants are used by related plans described in the next chapter, and it isn't necessary to understand these servants before using the plans. However, if you intend to create additional servants for use with the System Kit, it may be helpful to know that the System Kit defines, by example, types of servants identified by a file extension of `probes`, `probe`, and `node`. By modeling a servant after a `probes` servant, it can be used with the SystemThresholds compiler, or a compatible compiler. By modeling a servant after a `probe` servant, it can be used with the SystemThreshold compiler, or a compatible compiler. The SystemThreshold.node servant is a helper to both the SystemThreshold and SystemThresholds compiler.

When creating compilers and servants of your own, after first just getting something to work, consider how your might conform to an existing model or API for reusability, or how you might define a new one that others can follow. The idea is to build up a set of reusable compilers and model plans, so others can focus on the creation of plans built from the model, and servants that plug into those models.

8 Plans

Plans are XML documents that represent a set of NodeBrain rules, or configuration file, as a table with options. This kit provides just a few to get you started with monitoring a server or application.

8.1 Folders

Folder are provided to aid in navigation when managing plans using the NodeBrain Planner.

Plan	Purpose
SystemFolder	Enables navigation to all application plans provided by this kit. Parent is _Admin.

8.2 Model Plans

Plans in this section are used as a model for creating new plans. They are children of the _Model folder.

Plan	Purpose
SystemAudit	Audit a log file, watching for patterns matching regular expressions.
SystemThreshold	Create a new theshold plan to monitor values returned from one or more types of "probe" servant.
SystemThresholds	Create a new theshold plan to monitor multiple values returned by a single "probes" servant.

8.3 Application Plans

Plans in this section provide a minimal application that can be used to monitor the health of a server or application.

Plan	Purpose
System	Agent responsible for monitoring a server or application. If monitoring a NodeBrain application, additional agents are often used to implement the application.
SystemCpu	SystemThresholds plan that uses the SystemCpu.probes servant to monitor CPU utilization.
SystemFilesys	SystemThreshold plan that uses the SystemFilesys.probe servant to monitor file system space.
SystemProcess	SystemThreshold plan that used the SystemProcess.probe servant to monitor process to ensure required processes are running, and forbidden processes are not.

Index