Caboodle NodeBrain Kit

Release 0.8.17

Caboodle NodeBrain Kit
August 2014
NodeBrain Open Source Project

History

2007-11-05	Title: *Caboodle 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 • Minor corrections • Revisions to cover added and modified plan.

Preface

This document describes the Caboodle 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 Caboodle 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/OVO to schedule the execution of probes. These options have been removed from the Caboodle NodeBrain Kit, requiring a NodeBrain rule engine, although integration with commercial system management agents is still possible and often desirable.

You should expect bugs in releases prior to 1.0. Although this tool has been used successfuly for several years, defects have been forgiven and worked around. Changes have been made recently in preparation for the long promised release to the NodeBrain Open Source Project, potentially introducing new defects that have yet to be discovered. So keep your expectations low until a 1.0 release, and treat the prototype 0.x releases somewhat like a concept demonstration.

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 Caboodle NodeBrain Kit.

1.1 Caboodles

A NodeBrain caboodle is a directory that contains a NodeBrain application constructed from one or more NodeBrain kits. Kit components are designed to be invoked with a caboodle as the working directory, and reference all other components using a relative path. This means a system may host multiple instances of a given NodeBrain application in multiple caboodles without interference.

Suppose we have two caboodles, '/var/mycab1' and '/var/mycab2', representing either two instances of application A, or an instance of application A and an instance of application B.

```
/var/mycab1/
/var/mycab1/adapter/foobar.alarm
/var/mycab2/
/var/mycab2/adapter/foobar.alarm
```

Components of a caboodle must never reference other components of the caboodle using fully qualified names like '/var/mycab1/adapter/foobar.alarm'. Instead, they always assume the working directory is the caboodle and make a relative reference to other components. This makes them portable from one caboodle to another as illustrated by the following line of a shell script that could run in either of the caboodles shown above.

```
echo "big problem" | adapter/foobar.alarm
```

1.2 Subdirectories

Caboodles have a defined subdirectory structure upon which NodeBrain kits depend. Each subdirectory has a file management category.

Management	**Description**
Shared	Directory containing components that may be synchronized over multiple instances of a caboodle. Files in these directories may be placed under version control and checked out to different users or servers. For a server health monitoring application, as an example, each of several centrally managed servers may have a caboodle that has identical files in the shared management directories. An upgrade might be pushed through a common version control repository or a tar file.
Instance	Directory containing configuration components that make a caboodle unique from other caboodles within the same application. For a server health monitoring application, some configuration files must be unique to each server or type of server, and therefor should reside in an instance directory and should not be synchronized. Where possible, these components should be generated by shared components based on a minimum set of variables defined for each caboodle. Keep in mind that components that can adapt to the requirements of a given instance at load time (e.g. NodeBrain rules) can reside in a "shared" directory.
Dynamic	Directory containing files that are updated during the normal operation of a caboodle. Log files are an example. Unlike instance files that make a caboodle behave in a unique way from others within the same application, dynamic files are unique as the result of what happens in the caboodle. The application must manage these files to avoid unconstrained growth.

The file management category of each caboodle subdirectory is listed below with a short description of how the subdirectory is used.

Subdirectory	**Category and Description**
.nb	Instance: A special directory for use by the NodeBrain interpreter, nb. If a `caboodle.nb` file is found in this directory, NodeBrain sources it at startup. Variables defined here may be used within shared rule files to adapt to the unique requirements of a particular instance of an application caboodle. This is the only directory within a caboodle of which the NodeBrain interpreter seems to have some awareness. In fact, the NodeBrain interpreter has no awareness of caboodles. It simply supports a `.nb` directory in the working directory, and the caboodle concept depends on the caboodle being the working directory when the interpeter and all other components are invoked. This directory should not be used by components other than NodeBrain and node modules.
adapter	Shared: Commands that adapt a NodeBrain application component to specific interface requirements. For example, alarm adapters can be included here to adapt alarms to a particular protocol or to an external event management system interface.
agent	Shared: NodeBrain agent scripts. It is best to manage agents using an agent plan, in which case the agent script in this directory will be generated for you by the agent compiler. This directory also has files that identify the expected state of each agent.
bin	Shared: Used for commands that do not fall into the categories for which other directories are defined. See `adapter`, `agent`, `exit`, and `servant` directories.
cache	Dynamic: This directory has subdirectories that contain files used to cache information for a limited duration to avoid more expensive lookup, often where lookup may be relatively frequent. In cases where deleting the data would impact more than just the performance hit of having to perform a more expensive lookup, the `var` directory should be used instead.
config	Instance: This directoy may continue configuration files that are unique to a given caboodle. However, this directory is deprecated and you should use the `etc` directory for all new configuration files.
etc	Instance: Configuration files unique to a given caboodle amongst multiple caboodles synchronizing other components via a version control system. Since it is not appropriate to version control this file in a repository shared by other caboodles, it is sometimes helpful to manage these files using plans with compilers that can be directed to produce files in this directory based on symbolic variables and conditional compilation. See the `.nb` and `plan` directories.

exit	Shared: Commands that communicate a result via their exit code only. Unlike servants, these commands are normally not designed for use with NodeBrain. However, their exit code may identity the state of a monitored element, making it easy to map the exit code into a NodeBrain assertion.
kit	Shared: Used to manage kit dependencies. This directory is reserved for the nbkit command and should not be used by other kit components.
lib	Shared: Libraries and Perl modules supporting other kit components. As with all caboodle components, these components must be referenced using a relative path to maitain portability of the caboodle.
log	Dynamic: Log files for agents and other kit components. NodeBrain agents normally manage the rotation, compression, and deletion of their log files. When including other log files here, remember to provide the necessary management to avoid filling up your file system. Consider using the Caboodle agent to manage additional log files unless you have a more convenient method.
message	Dynamic: Used by the Message module and other modules using the NodeBrain Message Log API. Do not use this directory for other purposed.
msg	Shared: Used by alarm adapters (e.g. adapter/mail.alarm) that lookup and include a description of the alarm condition and response instructions. Subdirecties are defined for logical sets of alarms identified by the first three or four characters of a message identifier. For example a message CAB0001 would be described in the CAB subdirectory in a file named CAB0001. Within the message set subdirectory (e.g. CAB) additional files provide header, footer, and missing file text.
out	Dynamic: Output files produced by commands issued by NodeBrain when the output is not redirected. These files are retained for a few days for troubleshooting.
pipe	Dynamic: FIFO files for sending commands to Pipe nodes in NodeBrain agents or for communicating between other caboodle components.
plan	Shared: Contains a subdirectory for each plan, including the XML document, plan relationship files, and normally any files generated from the XML document by the associated compiler. In some cases the generated files go to another directory (e.g. `etc`) that does not share the same version control repository used by the plan directory.
queue	Dynamic: This directory contains subdirectories for Peer module gueues. This mechanism is deprecated and a future release of the Peer module will switch to message logs. NodeBrain message logs provide a more efficient method of sending events between agents and are currently supported by the Message module. See the Message module manual for more info.
security	Instance: Contains certificates and access lists.

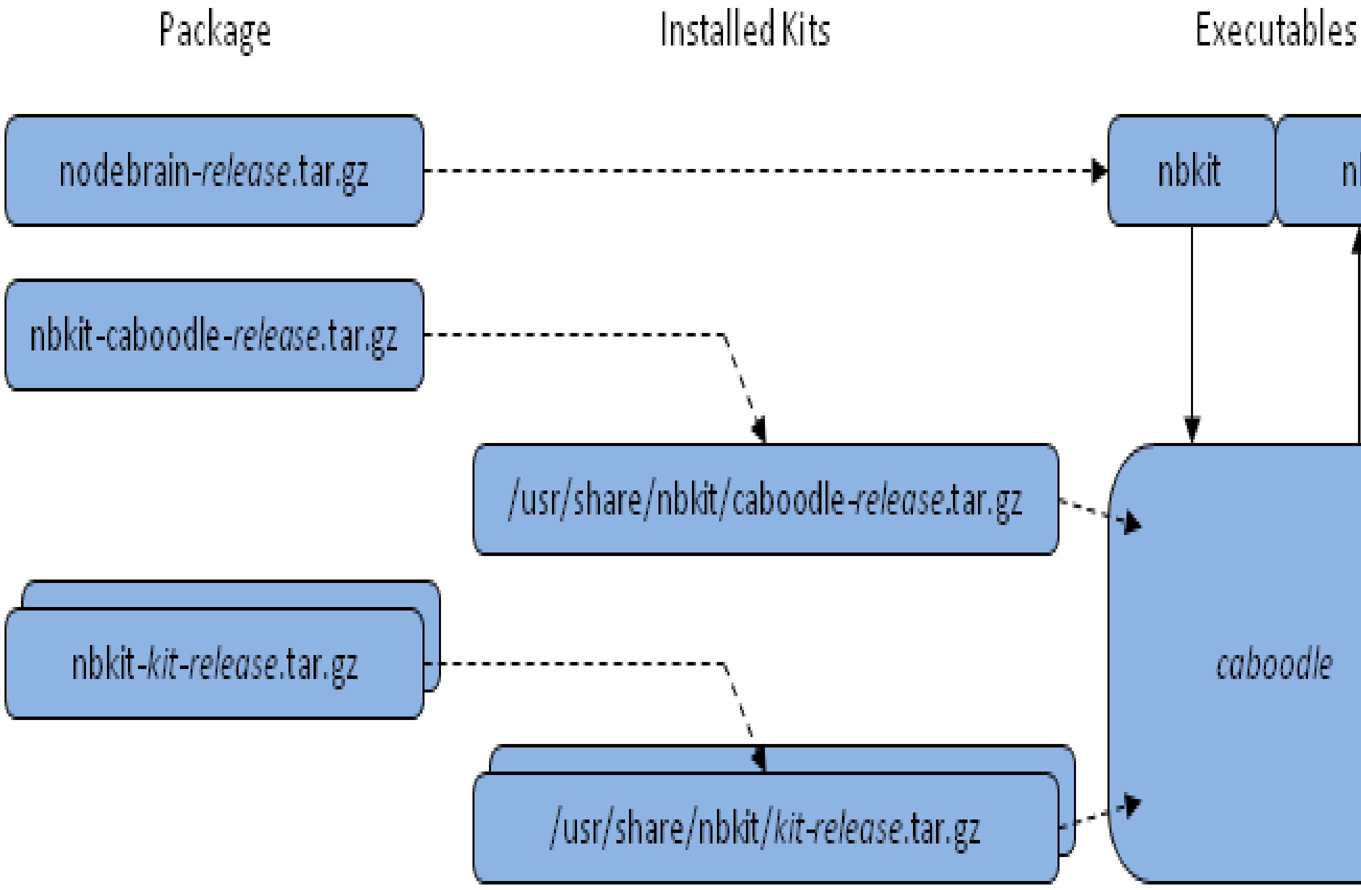
servant	Shared: Contains servant commands that output NodeBrain commands to stdout. Commands that do not conform to this standard belong in bin, adapter, or exit. Servant commands may optionally accept commands from NodeBrain on stdin and report error conditions on stderr.
setup	Shared: Setup components. A kit may provide components in this directory for setting up an application.
socket	Dynamic: Local domain socket files for communication between NodeBrain processes using the Peer module. This directory may also be used for communication between other caboodle components that use local domain socket files.
user	Shared: Application user files. This directory is for storing user preferences and other user related information. There is no recommendation on how to organized user data in this directory.
var	Dynamic: Data files used in the normal operation of a NodeBrain application. Use the `cache` directory for temporary caching of data obtained from a more authoritive source.
web	Shared: Web content and scripts for web based tools. Individual tools should use sub-directories. For example, the NodeBrain planner uses the `"planner"` subdirectory and the Webster modules uses the `"webster"` subdirectory by default.

1.3 Kits

A NodeBrain kit is a collection of components used to construct a NodeBrain application caboodle. Kits are normally installed to '`/usr/share/nbkit`' or '`/usr/local/share/nbkit`' on development servers where application caboodles are constructed. Kits are not required on test and production servers since their components migrate as part of the application caboodle.

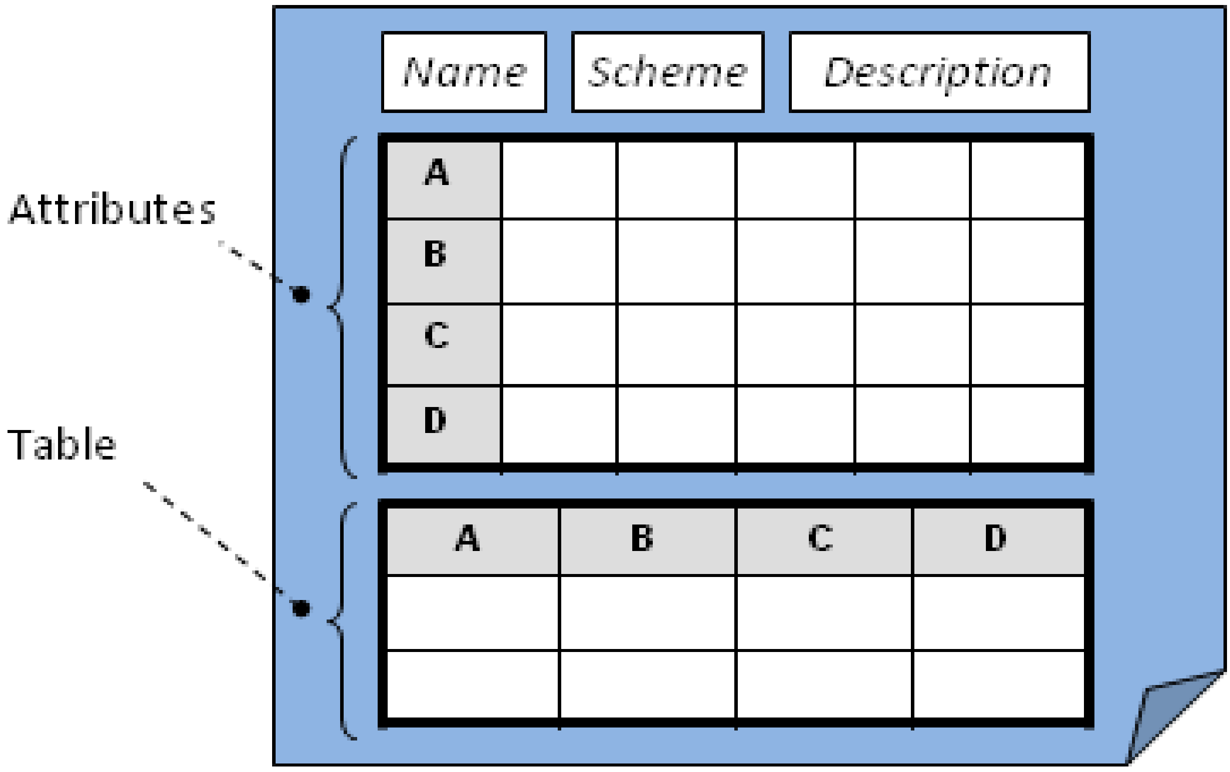
```
/usr/share/nbkit/kit-release.tar.gz
/usr/share/nbkit/caboodle-0.8.14.tar.gz
```

An installed kit archive file is extracted into a caboodle when the a developer selects the kit for use. Within a caboodle, kits are stored in the '`kit/use`' subdirectory. These directories have the same structure as the caboodle itself, but are only referenced for the purpose of constructing the caboodle, they are not involved in application execution.

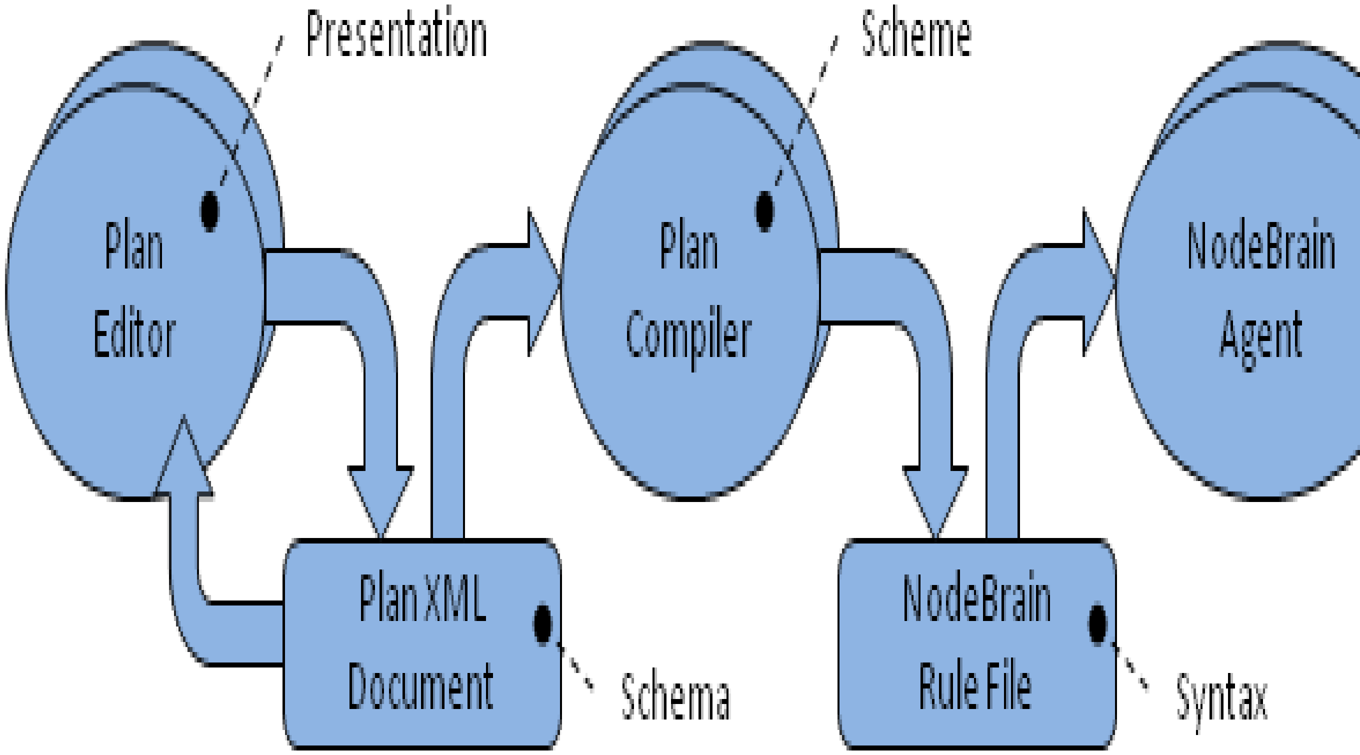

Package
Installed Kits
Executables
nodebrain-*release*.tar.gz
nbkit
nbkit-caboodle-*release*.tar.gz
/usr/share/nbkit/caboodle-*release*.tar.gz
nbkit-*kit-release*.tar.gz
caboodle
/usr/share/nbkit/*kit-release*.tar.gz

1.4 Plans

A *plan* is an abstraction of a set of NodeBrain rules, managed as an XML document using an XML schema that supports self-defining tables. The attributes of a plan have properties that identify how they are used. Attributes may be used as columns of the table, or may be used as single value plan options.



A plan editor is used to update a plan XML document. A plan compiler translates the XML document, following a predefined *scheme* identified within the document, into NodeBrain rules or configuration files for other components making up an application.

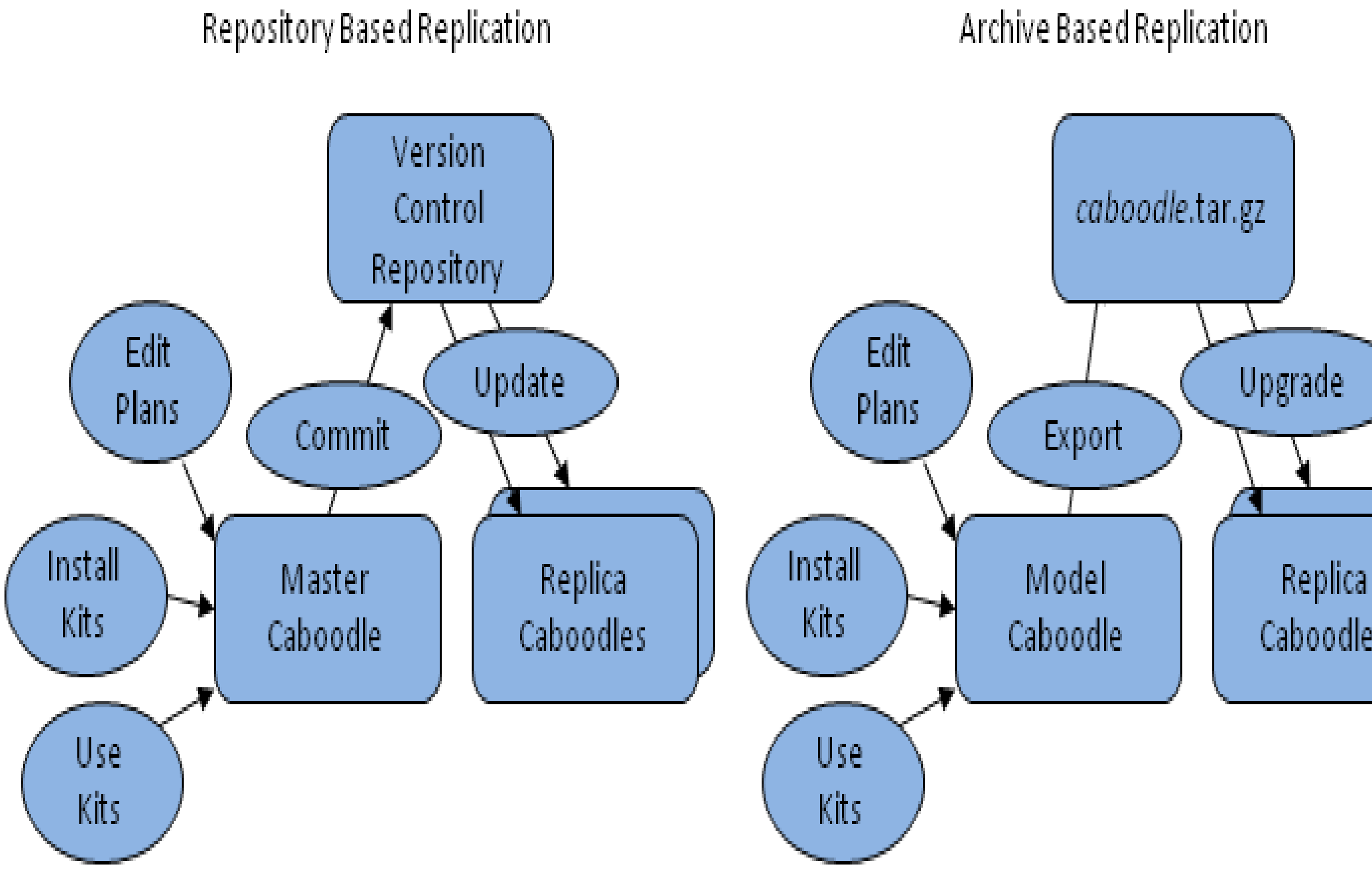


1.5 Component Replication

It is common in monitoring applications to have multiple servers that require the same monitoring rules and supporting scripts. The NodeBrain agent is willing to talk to peers, but the concept of a management server does not exist. In situations where it is necessary to centrally manage multiple instances of an application caboodle, there are two basic models to consider.

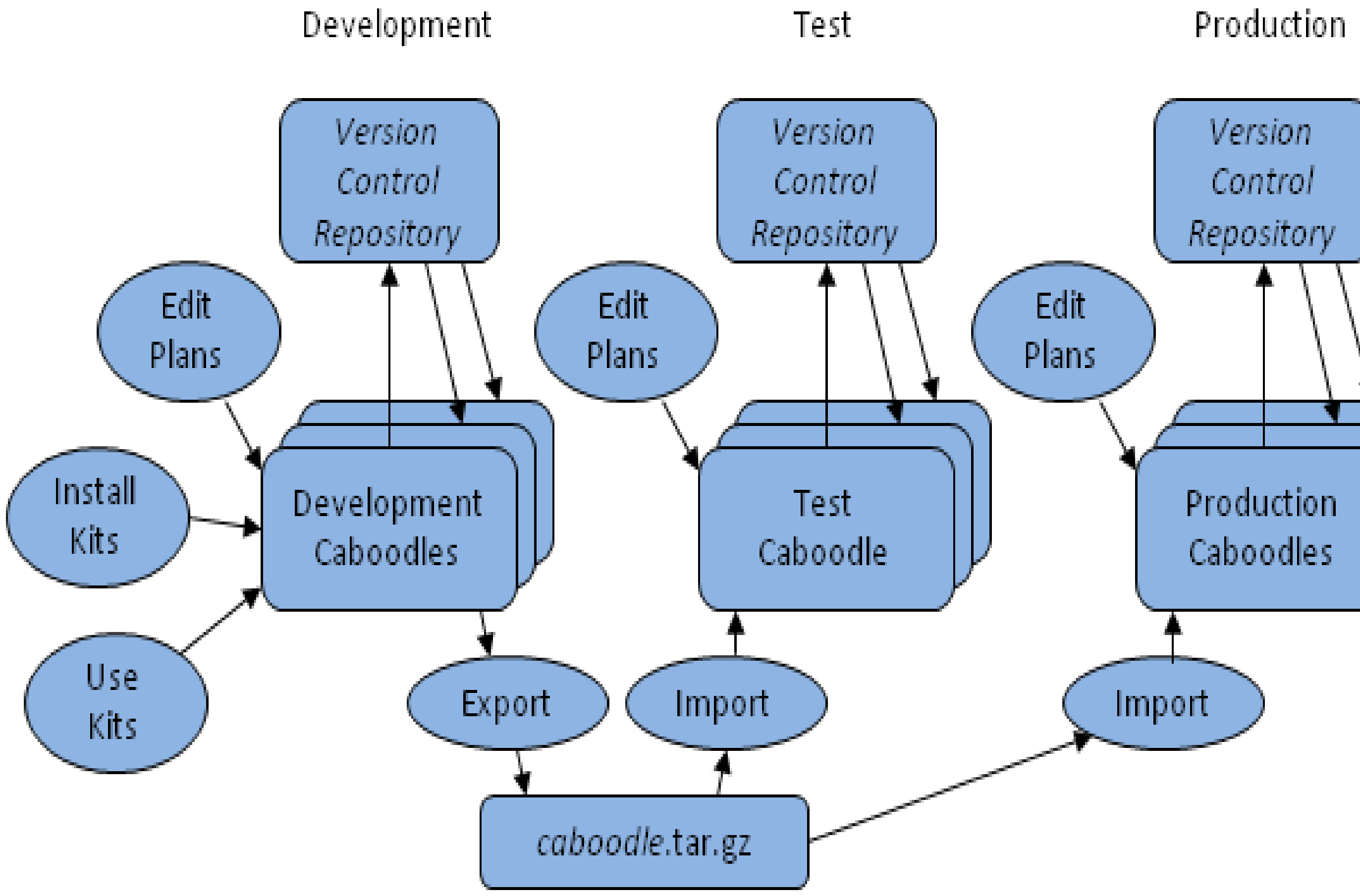
The first method makes use of a version control repository, where each instance of the application caboodle has a checked out copy of the components you want to replicate. A master caboodle is used for applying changes via a commit and each replica performs an update to get the changes. The replicas need only read access to the repository, while the master needs update permission. This approach can be combined with a web based plan editor for relatively quick distribution of rule changes.

The second method uses a simple tarball to transport updates. This approach is fine when changes are less frequent, and may be required when there is an administrative air gap that rules out the repository option. The export and import functions are provided by the Caboodle Kit to limit replication to components based on the management category of each directory, and to give plans special attention.



1.6 Component Migration

The tarball method of replication may be used for component migration through environments, perhaps combined with the repository method for replication within an environment. The export and import operations take special care in handling plans. Plan permissions are managed as separate XML documents called *control* plans, that only import when new. So permissions to modify plans do not replicate via the tarball method, even when the associated plan document does. In addition, only new plans and plans not previously modified in the target system will import. This makes it possible to elect to manage a plan in a given environment and not to accept updates via migration. This does not prevent migration further down stream, since this is a down stream decision. Plans that are not imported are actually temporarily stored in a kit/plan directory where they are available for manual analysis.



1.7 Component Naming Standard

To avoid name collisions in caboodle subdirectories, components and lower level subdirectory names are expected to be in CamelCase, starting with the name of the kit or application that provides the component. For example, components provided by the System NodeBrain Kit start with "System". Components provided by the Caboodle kit violate this standard a bit, but in a way that will not collide with kits following the standard. The caboodle kit provides components associated witht the Caboodle agent, which start with "Caboodle". A more foundational set of nbkit components start with an underscore ("_"), "nbkit", "NBK", or names starting with lower case. You should not emulate these deviations from the standard. Stick with CamelCase names starting with a short identifier of your kit or application.

2 Installation

The Caboodle 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-caboodle-0.8.14.tar.gz
# cd nbkit-caboodle-0.8.14
# ./configure
# make
# make check
# make install
```

When using this method, the kit is installed to '`/usr/local/share/nbkit/caboodle-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/caboodle-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-caboodle-0.8.14-1.noarch.rpm
```

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

3 Setup

This chapter walks the reader through the initial caboodle setup process as performed on a development system. After describing the individual steps, a complete example is provided in the last section of this chapter. The `nbkit` commands used in this chapter are explained in more detail in the next chapter, *Commands*.

3.1 Create Account

A caboodle can operate under any account on a system, although it is often best to run a NodeBrain application using an application account. When using an application account, an administrator with `root` would create the application account first.

```
# useradd user
```

You can skip this step if creating a caboodle to run under your personal account.

3.2 Create Directory

Create a directory you will use as your caboodle. If creating an application caboodle as `root`, also assign ownership to the application account. Otherwise, you can skip the `chown` command.

```
# mkdir directory
# chown user:user directory
```

3.3 Create Link

Give your caboodle a name to use with the `nbkit` command.

```
$ nbkit caboodle link directory
```

3.4 Initialize Caboodle

Specify the kit you want to use for the caboodle. There may be multiple versions of the Caboodle NodeBrain Kit installed on the server in various locations, so you need to specify the full path. Normally NodeBrain kits will be installed at '`/usr/share/nbkit/`' or '`/usr/local/share/nbkit/`'. For example, a normal install of release 0.8.14 of the Caboodle NodeBrain Kit using an RPM will install it as '`/usr/share/nbkit/caboodle-0.8.14.tar.gz`'.

```
$ nbkit caboodle use kit
```

3.5 Create Caboodle Agent Identity

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

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

3.6 Start Caboodle Agent

Use the following command to start the Caboodle agent. This agent is responsible for alarm distribution and removing some temporary files from the caboodle.

```
$ nbkit caboodle start Caboodle
```

3.7 Edit CaboodleMailAdmin Plan

The `CaboodleMailAdmin` plan subscribes to all alarms initially, and sends email to the local root account. If that is not appropriate for the caboodle you are setting up, change the administrator email address as follows.

```
$ nbkit caboodle edit CaboodleMailAdmin
:%s/root@localhost/email-address/g
:wq
$
```

3.8 Issue Test Alarm

To verify that alarms are properly sent to the configured email address, issue a test alarm.

```
$ nbkit caboodle alarm text="CAB0000 This is only a test"
```

3.9 Setup Example

All the steps described above are illustrated by an example here. This example assumes an application account and caboodle is being created by `root`.

```
# useradd cabby
# mkdir /var/cabby
# chown cabby:cabby /var/cabby
# su cabby
$ nbkit cabby link /var/cabby
$ nbkit cabby use /usr/share/nbkit/caboodle-0.8.14.tar.gz
$ nb
> peer.identify Caboodle
> quit
$ echo "declare Caboodle identity;" >> ~/.nb/user.nb
$ nbkit cabby start Caboodle
$ nbkit cabby edit CaboodleMailAdmin
:%s/root@localhost/john.p.doe@acme.com/g
:wq
$ nbkit cabby alarm text="CAB0000 This is only a test"
```

Here's another example, assuming a user is setting up a caboodle on an existing account that already has other caboodles.

```
$ mkdir ~/cabby
$ nbkit cabby link $HOME/cabby
$ nbkit cabby use /usr/share/nbkit/caboodle-0.8.14.tar.gz
$ nbkit cabby start Caboodle
$ nbkit cabby edit CaboodleMailAdmin
:%s/root@localhost/john.p.doe@acme.com/g
:wq
$ nbkit cabby alarm text="CAB0000 This is only a test"
```

4 Commands

The `nbkit` command, an alias for the NodeBrain interpreter `nb`, is used to launch Perl scripts that reside in the '`bin`' subdirectory of a caboodle. Parameters to `nbkit` enable operations on a caboodle and individual agents and plans defined within the caboodle. The general syntax is as follows.

```
$ nbkit caboodle verb arguments
```

In addition to this command line interface (CLI), the Caboodle NodeBrain Kit provides a web interface supporting many of the operations described in this section.

4.1 Caboodle Links

Caboodle links are used to associate a caboodle directory with a short name for use in `nbkit` commands. The commands in this section are used to manage these links, which are actually symbolic links within the user's '`~/.nb/caboodle/`' directory.

4.1.1 link

The caboodle is given a short name `nbkit` with a verb of `link`. In the following example the name `cab` is defined as a link to the directory created above. This definition applies only to the account that performs the *link* command.

```
$ nbkit cab link $HOME/foobar
```

4.1.2 unlink

If at any time you decide to abandon a caboodle, use the `unlink` command.

```
$ nbkit cab unlink
```

4.1.3 –caboodles

All caboodles currently defined for your account can be displayed with either of the following commands.

```
$ nbkit -c
$ nbkit --caboodles
```

4.2 Caboodle Development

On a development system where the Caboodle NodeBrain Kit is installed, you start a caboodle by creating an empty directory and giving it a short single word name using the `link` command. In the following example, the caboodle directory is created as '`foobar`' in the user's home directory, but it can be anywhere.

```
$ cd ~
$ mkdir foobar
$ nbkit cab link $HOME/foobar
```

4.2.1 –kits

To list installed NodeBrain kits, use one of the following commands.

```
$ nbkit -k
$ nbkit --kits
```

4.2.2 use

Now it is time to populate the caboodle with components provided by NodeBrain kits, starting with the Caboodle Kit. The following example assumes release 0.8.14, but you will adjust the release number based on the installed kits available. Because the Caboodle Kit provides the logic for most nbkit verbs, including `use`,

```
$ nbkit caboodle use kit-archive
$ nbkit cab use /usr/share/nbkit/caboodle-0.8.14.tar.gz
```

4.3 Component Migration

The `export`, `import`, and `upgrade` commands may be used to migrate caboodle components.

4.3.1 export

The export command is used to create an archive of caboodle components in the "shared" management category. This is done to create a backup, or to enable transport to another caboodle, perhaps on another server. Log files and other files considered to be caboodle specific and of no value to similar caboodles are not exported.

```
$ nbkit caboodle export caboodle-archive
$ nbkit cab export /tmp/foobar-1.0.9.tar.gz
```

4.3.2 import

Use the `import` command to update a caboodle from an archive file produced by the `export` command. This command will add new "shared" files found on the archive, and will not disturb files in the caboodle that are not included in the archive. An import takes additional care in handling plan files, since they may be uniquely managed in an environment. A plan that has local modifications is not updated from the archive, a message is displayed, and the archived copy is placed in a kit/plan/*plan* directory for manual analysis if necessary. This special handling is not applied to other types of components, only plans.

```
$ nbkit caboodle import archive_file
$ nbkit cab2 import /tmp/foobar-1.0.9.tar.gz
```

4.3.3 upgrade

Use the `upgrade` command to update a caboodle from an archive file produced by the `export` command, including the removal of "shared" files not found in the archive. This command is used to manage copies of a caboodle, without allowing for local modifications.

```
$ nbkit caboodle upgrade archive_file
$ nbkit cab2 upgrade /tmp/foobar-1.0.9.tar.gz
```

4.4 Agent Management

Commands in this section operate on NodeBrain agents. An agent is a NodeBrain script that runs as a daemon.

4.4.1 archive

The NodeBrain interpreter running in agent mode responds to an `archive` command by renaming the current log file by inserting a time stamp, and then starting a new log file under the original name. For example, an agent named "Foobar" would normally have a log file named ‘`log/Foobar.log`’. An `archive` command renames it to ‘`log/Foobar.YYYYMMDDHHMMSS.log`’, where YYYYMMDDHHMMSS is year, month, day, hour, minute, and second at the time of the command.

An agent normally has a rule that issues an `archive` command daily. However, the `archive` verb of the `nbkit` command may be used to send an `archive` command to an agent at other times. This may be useful when troubleshooting if the log file has become too large to view.

```
$ nbkit caboodle archive agent
$ nbkit cab archive Caboodle
```

4.4.2 bounce

The `bounce` command is used to stop and restart an agent. This may be done to reload rules after modification. If the current expected state is set to "up", just like a `start` command.

```
$ nbkit caboodle bounce agent
$ nbkit cab bounce Caboodle
```

4.4.3 check

The `check` command compares the current state of an agent with the expected state. If the agent is "down" and expected "up", it is started. If the agent is "up" and expected "down", it is stopped.

```
$ nbkit caboodle check agent
$ nbkit cab check Caboodle
```

4.4.4 connect

The `connect` command is used to invoke the NodeBrain interpreter in interactive mode and prompt for commands to be issued to an agent. This requires that the agent have a Peer server node named the same as the agent. This is normal for agents generated by agent plans.

```
$ nbkit caboodle connect agent
$ nbkit cab connect Caboodle
```

4.4.5 start

The `start` command is used to start an agent that is currently "down" and set the expected state to "up".

```
$ nbkit caboodle start agent
$ nbkit cab start Caboodle
```

4.4.6 stop

The `stop` command is used to stop an agent that is currently "up" and set the expected state to "down".

```
$ nbkit caboodle stop agent
$ nbkit cab stop Caboodle
```

4.5 Plan Management

Commands in this section operate on NodeBrain plans. A plan is an XML document that normally translates into a set of NodeBrain rules or a configuration file based on a model. Compilers are provided to perform this translation. Each plan has a "scheme" that determines which compiler is used.

Most operations performed on plans by commands in this section may also be performed by the NodeBrain Planner web interface.

4.5.1 compile

The `compile` command is used to convert a plan XML document into a target format, often NodeBrain rules.

```
$ nbkit caboodle compile plan
$ nbkit cab compile Caboodle
```

4.5.2 disable

Use the `disable` command to disable a plan, or disable a relationship between two plans.

```
$ nbkit caboodle disable plan
$ nbkit cab disable CaboodleMailAdmin
```

```
$ nbkit caboodle disable plan.other.relationship
$ nbkit cab disable CaboodleMailAdmin.CabaoodleNotify.parent
```

4.5.3 edit

The `edit` command is used to modify a plan. The XML document is first converted into a temporary file with a less complex format and vi is invoked to edit the file. After exiting vi, the file is converted back into XML document format. The XML document is then converted into a target format by the associated compiler.

```
$ nbkit caboodle edit plan
$ nbkit cab edit Caboodle
```

4.5.4 enable

The `enable` command is used to enable a plan, or enable a relationship between two plans.

```
$ nbkit caboodle enable plan
$ nbkit cab enable CaboodleMailAdmin
```

```
$ nbkit caboodle enable plan.other.relationship
$ nbkit cab enable CaboodleMailAdmin.CaboodleNotify.parent
```

4.5.5 list

Use `list` to display a list of configured plans.

```
$ nbkit caboodle list
$ nbkit cab list
```

4.5.6 remove

The `remove` command is used to remove a plan for a caboodle.

```
$ nbkit caboodle remove plan
$ nbkit cab remove Foobar
```

4.5.7 rename

The `rename` command is used to change the name of an existing plan.

```
$ nbkit caboodle compile plan.newname
$ nbkit cab rename Caboodle.Cab
```

4.5.8 setup

The `setup` command may be used to display a list of plans available for setting up in the caboodle.

```
$ nbkit caboodle setup
$ nbkit cab setup
```

This command may also be used to invoke the assisted setup script for a plan.

```
$ nbkit caboodle setup plan
$ nbkit cab setup Foobar
```

4.5.9 show

The `show` command displays information about a plan, include the scheme title, and relationships to other plans.

```
$ nbkit caboodle show plan
$ nbkit cab show Caboodle
```

4.5.10 view

Use `view` to browse a plan in configuration file format.

```
$ nbkit caboodle view plan
$ nbkit cab view Caboodle
```

```
|
```

4.6 Alarm Distribution

Alarm distribution is handled by the Caboodle agent, and required one or more alarm subscription rules. The CaboodleMailAdmin plan is an example of an alarm subscription. This plan can be modified to send email to your email address using the following command. By default the email address is root@localhost. Change it to your email address.

```
$ nbkit caboodle edit CaboodleMailAdmin
```

When you save the change, the Caboodle agent should automatically stop and restart, unless it is in an expected down state. Before experimenting with the `alarm` command in the next section, make sure the Caboodle agent is up by issuing a `bounce` command.

```
$ nbkit caboodle bounce Caboodle
```

4.6.1 Alarm

The `alarm` command sends a NodeBrain alert to the alarm node of the Caboodle agent supporting the specified caboodle.

```
$ nbkit caboodle alarm attribute="value" [ ... ]
```

Multiple optional alarm attributes may be specified. Some have default values when not specified. All alarm attributes may be referenced by an alarm subscription, which is required for actual distribution of the alarm.

Attribute	**Description**
appl	Specify the name of the application to which the alarm applies.
group	Specify a value that categories the alarm; e.g. "OS", "Database", "Web". The group attribute is intented to route alarms to a unique group of subject experts. This is unlike the route attributes which may be used to provide visibility to various inter- ested recipients.
node	Specify the hostname associated with the alarm. By default, the local hostname is used.
route	Specify a comma seperated list of routing codes for alarm subscriptions. The codes are strings that have meaning within a given application. For example, if the caboodle is a security event monitoring application, a value of "SOC" might be used to indicate the alarm is intented for the Security Operations Center. Multiple codes may be entered; e.g. "SOC,NOC".
severity	Specify a level of severity for the alarm. Typical values are "critical", "major", "minor", "warning" and "info", although you may select alternate values that work bet- ter in the context of your application.
text	Specify message text that describes the alarm condition; e.g. "Roof flew off the barn". You may include a message identifier of the form AAANNNN at the start of the text; e.g. "PWS0125 Roof flew off the barn".

5 Agents

This section describes agents provided by this kit. Although there is only one agent for this kit, and not much to say about it, the structure of this document is intended as an example for other kits or applications, some of which will have several agents.

5.1 Caboodle Agent

This kit provides an agent called Caboodle, which is responsible for alarm distribution, and may also optionally host web tools like the NodeBrain Planner described later in this document.

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.

The Caboodle NodeBrain Kit provides a small set of adapters to get started. Other kits can provide additional adapters. You are advised to think of your application as a kit, even if you don't publish it, and name components using CamelCase, starting with a single word identifier for your application or kit. This is important because caboodle components are stored in a flat name space where collisions would be likely without following this naming standard.

Adapters are stored in the '`adapter`' subdirectory of a caboodle and have extensions that identify an adapter type.

6.1 Plan Compilers

Each plan is associated with a *scheme*, which is a model for converting a table represented as XML, into a set of NodeBrain rules, or a configuration file for another tool. Compilers are adaters that perform this conversion for a given scheme, or you might say they implement a scheme.

These components are named *scheme*.compiler. The quickest path to understanding a compiler is to study a plan using the scheme. For example, the Caboodle plan uses the _Agent scheme. Looking at both the plan XML document and the generated '`*.nb`' file reveals the transformation. If you are comfortable with Perl, a peek at the compiler itself will clarify further.

```
_Agent.compiler
_Alarm.compiler
_Alarmer.compiler
_Block.compiler
_Config.compiler
_Distribution.compiler
_List.compiler
_Morph.compiler
_Node.compiler
_Package.compiler
_Permissions.compiler
_Policy.compiler
_Rule.compiler
_Subscription.compiler
_Table.compiler
_Translator.compiler
```

6.2 Alarm Adapters

Alarm adapters are used to match alarms to specific methods of delivery and enable a variety of formats for a given delivery method. Although most applications need to send alarms via a variety of the methods, such as SMTP, SNMP traps, syslog, and command interfaces to event management or ticketing systems, the initial release of this kit only provides an adapter for mail.

```
_Mail.alarm
_Mail.form
```

Plans that use the _Subscription compiler allow for the specification of a `form` and `alarm` adapter. By addding more `form` and `alarm` adapters you increase the options for alarm subscriptions.

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.

Although servants are critical to many NodeBrain applications, the initial release of the *Caboodle NodeBrain Kit* provides no servants. See the *System NodeBrain Kit* for examples.

8 Plans

Plans are XML documents that represent a set of NodeBrain rules, or other configuration files, as a table with options. This kit provides two sets of plans. The first set have names starting with an undercore ("_") and provide an initial folder structure and plans used as models for creating new plans. The second set have names starting with "Caboodle" and implement the actual application provided by this kit. Other kits and applications are expected to name both model and application plans in CamelCase starting with the name of the kit or application. The use of ("_") is unique to the Caboodle Kit.

8.1 Folders

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

Plan	**Purpose**
_Home	Top level folder.
_Admin	Plan of interest primarily to application administrators.
_Admin/_Model	Plans used as models for new plans. Any plan can server as a starting point for creating a new plan. However, plans in the _Model folder typically have no other purpose. These plans are included in a pull down menu of model plans when creating a new plan using the NodeBrain Planner web interface.
_Admin/_Schema	Examples of supported XML schemas (Folder and Table).
_Admin/_Scheme	Model for each supported scheme for which a compiler is provided. This is a proper subset of the plans in the _Model folder.
_Admin/CaboodleKit	Enables navigation to all application plans provided by this kit.
_Alarms	Plans that generate an alarm identified by the plan name. This is intended as a single place to find all alarms, although alarms will be found under other folders and plans as well.
_Planner	Plans of interest to users called "planners", who are responsible for updating the content of plans, but not the design of plans.
_Subscriptions	Alarm subscription plans.

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**
_Agent	Create an agent. This plan is like a _Block plan, but volunteers some addition rules appropriate for all agents.
_Alarm	Create an alarm. Works with the CaboodleAlarmer plan to generate a flood protected alarms.
_Alarmist	Create a mulitple condition alarm. Works with the CaboodleAlarmer plan to generate a flood protected alarm covering multiple conditions, each identified by a signature name.

_Block	Create a set of rules organized into blocks that can be enabled or disabled. These rules are represented in NodeBrain rule syntax.
_Config	Create a configuration file for a non-NodeBrain application. This is like a _Block plan, in that it has blocks that can be enabled and disabled, but it also supports block conditions interpreted by NodeBrain for selective inclusion of blocks in the generated file. For NodeBrain rules, the _Block plan should be used instead, because the conditions can be specified within the NodeBrain syntax and applied at agent startup instead of plan compilation time.
_Folder	Create a new folder plan. Plans are stored in a flat namespace, unlike a multi-level directory system. All plans may have multiple parent plans and multiple child plans. This means every plan is effectively a folder of child plans, but a _Folder plan has no other purpose than to provide access to a set of child plans.
_List	Create a list of assertions for a node using a module that accepts assertions of tuples (e.g. Tree, and Cache).
_Macro	Create a macro for use by other plans.
_Morph	Create a non-NodeBrain application configuration file using NodeBrain conditions and symbolic substitution. These plans are used similar to _Config plans, but where symbolic substitution is required in addition to conditional use of blocks.
_Node	Create a node.
_Package	Create a package of plans with control of source parameters and the order children are sourced.
_Policy	Create a set of rules.
_Rule	Create a single rule.
_Subscription	Create an alarm subscription.
_Table	Create a plan using the table schema without generating code. A table plan may be used as a configuration file to an application component that accesses the XML document directly. A table plan can also be used to prepare a model plan for a new scheme. After formulating the model plan a compiler can be written and the plan scheme can be changed to use the new compiler.
_Translator	Create a translator node and translator. The translator may be used by other nodes as well.
_TypeList	Create a list of single attribute turple assertions (see _List above) where the attibute is called `"Type"`. This is a simple example of a _List plan, provided because use of _List requires modification of the plan table attributes to adapt to your requirement. The _TypeList plan is an example of a very minor modification.

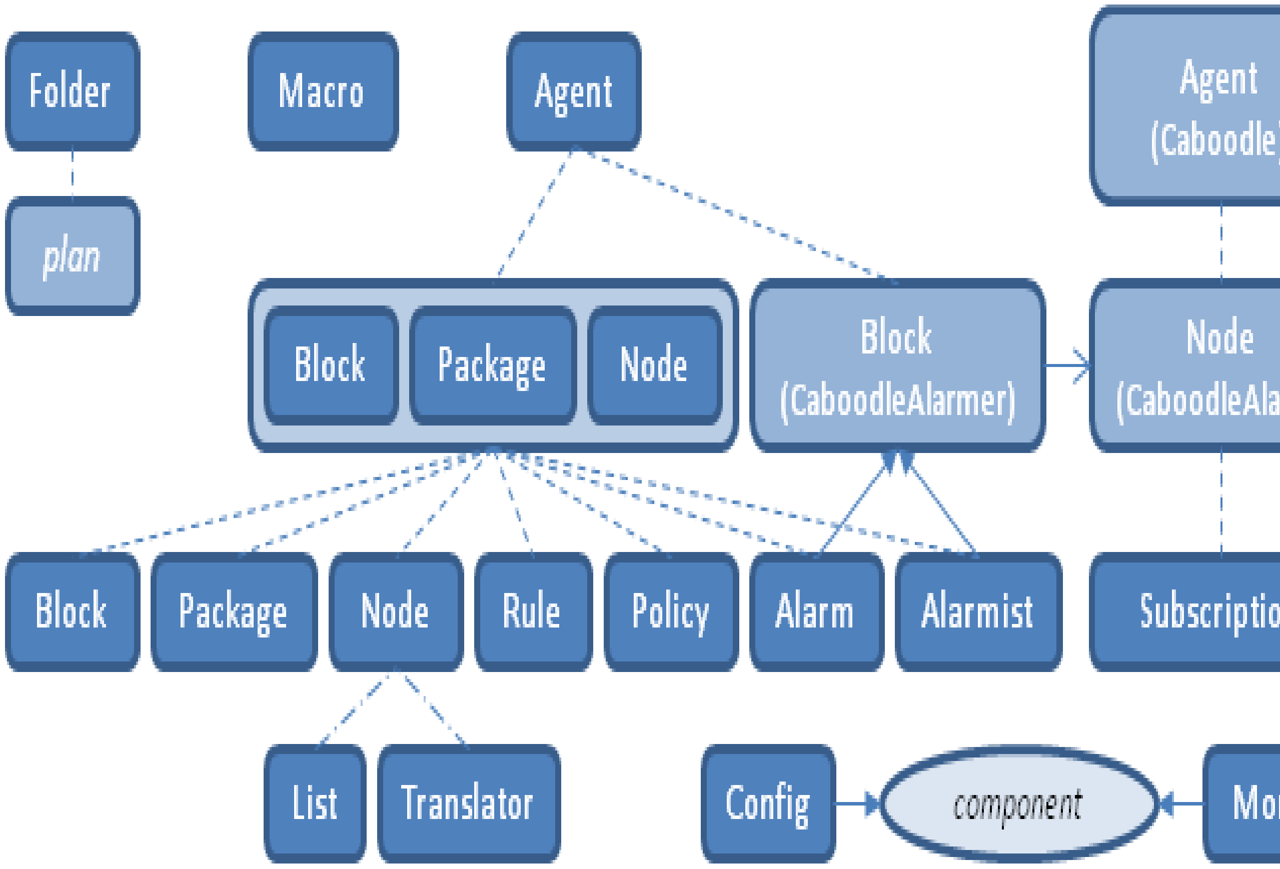
8.3 Application Plans

Plans in this section provide a minimal application that can be used to manage alarm distribution and an optional web interface for administrators.

Plan	**Purpose**
Caboodle	Agent responsible for alarm distribution and hosting of an option web server for tools
CaboodleAccess	User permissions for access to the optional NodeBrain Planner
CaboodleAlarm	Alarm distribution plan used by Caboodle agent to implement alarm subscriptions
CaboodleAlarmAttributes	Alarm attributes managed separately to symplify replacing CaboodleAlarm
CaboodleAlarmDelivery	Alarm delivery macro used by alarm subscriptions
CaboodleAlarmer	Alarm handler with flood protection
CaboodleConfig	Configuration of web heading for the optional NodeBrain Planner
CaboodleFilter	Access filter for the NodeBrain Webster node module when used as a web server
CaboodleMailAdmin	Alarm subscription for caboodle administrator
CaboodleProfile	Optional plan for managing a caboodle profile (.nb/caboodle.nb)

8.4 Plan Relationships

The figure below shows how plans can be organized based on their scheme. An agent plan is the top level plan for an agent, while a batch or interactive script would start with a block, package, or node plan. A node plan provides a single node, while a block or package plan can provide multiple nodes. Children of these types of plans are listed below them. List and translator plans are often referenced by a node directly instead of being sourced in as a child plan. Alarm and alarmist plans provide a method of generating alarms to be handled by CaboodleAlarmer which provides flood control and an interface to the Caboodle agent for distribution based on subscriptions. Morph and config plans normally stand alone because they generate configuration files for component other than NodeBrain. Macro plans are often managed as a resource loaded by the `%use` directive, so it is shown standing alone at at the top level. However, macros may also be defined within a lower level context, and may be a child to another plan.



9 Planner

The NodeBrain Planner is a web interface for editing plan documents in a caboodle, as a way of managing NodeBrain rules. This interface is provided by a Perl CGI script that you can use with an Apache web server, or with the Webster module provided with NodeBrain. Depending on your application, it may be best only to use this interface on a development platform. However, some applications require dynamic rule changes in a production environment, in which case you may elect to use this inferface in production as well as development.

Use of the NodeBrain Planner is optional. You may elect to manage a caboodle using the `nbkit` command at a shell prompt, or use the Planner and `nbkit` command in combination.

9.1 Certificate Authentication

Because the NodeBrain Planner enables control of a caboodle and the account running the Planner, it is important to authenticate administrators using the Planner and set appropriate permissions. Authentication is performed by the web server and web browser using X509 certificates and mutual verfication.

Both client and server require a certificate, private key, and at least one trusted cerfiicate that has signed the certificate of the peer. On the server side, these files can be stored in the '`security`' subdirectory of your caboodle as follows.

```
security/ServerCertificate.pem
security/ServerKey.pem
security/TrustedCertificates.pem
```

You can obtain the certificates from a trusted certificate authority or create your own using openssl. The is plenty of information on the web to reference. If you have a copy of the NodeBrain source package, you can also start with the Webster NodeBrain Module tutorial.

9.2 Authorization

The `CaboodleAccess` plan is used to control Planner permissions. To make yourself an administrator, edit the plan by issuing the following `edit` command.

```
$ nbkit caboodle edit CaboodleAccess
```

This will drop you into `vi` on the plan, after first converting it from XML format to a more `vi` friendly format called `"cfg"`. Once you get the web interface working, you will be able to use it for subsequent changes to the `CaboodleAccess` plan.

In `"cfg"` format, table rows are expressed in blocks starting with `"]"` in column one, and attribute names starting in column two, followed by a colon, `":"`, and a value the first two blocks shown below, are the last two blocks of the file when you first edit it. Copy the first of these to make a third as the bottom of the file as illustrated. The UserId must match the common name (CN) in your browser certificate, and the Role must be `"admin"` to give you full control of all the plans in the caboodle.

```
]
 UserId: admin
 Role: admin
 Name: Administrator
 Email: root@localhost
 Phone: Unknown
 Department: Unknown
 Location: Unknown
]
 UserId: guest
 Role: guest
 Name: Guest
 Email: root@localhost
 Phone: Unknown
 Department: Unknown
 Location: Unknown
]
 UserId: certificate-CN
 Role: admin
 Name: your-name
 Email: your-email-address
 Phone: Unknown
 Department: Unknown
 Location: Unknown
```

When you `:wq` out of `vi` reply `y` when asked if you want to compile the plan.

9.3 Apache

If you are familiar with the Apache web server, you may prefer to run the NodeBrain Planner under Apache. An outline for creating a virtual host for a caboodle is shown below. Replace *Caboodle* with your caboodle directory. You may use a different port number, and may place the SSL certificate and key files in a different location under different names. The important elements of this outline to follow are the `DocumentRoot`, the `Include` to control access, and the options related to CGI scripts.

```
Listen 443
<VirtualHost *:443>
DocumentRoot "Caboodle/web"
...
SSLEngine on
SSLCertificateFile Caboodle/security/ServerCertificate.pem
SSLCertificateKeyFile Caboodle/security/ServerKey.pem
SSLCertificateChainFile Caboodle/security/TrustedCertificates.pem
...
<Location />
SSLRequireSSL
Include Caboodle/plan/CaboodleAccess/CaboodleAccess-apache.txt
</Location>
...
<Files ~ "\.cgi$">
  SSLOptions +StdEnvVars
</Files>
...
<Directory "Caboodle/web">
  Options ExecCGI ...
  ...
</Directory>
...
</VirtualHost>
```

The NodeBrain Planner must run as the user owning the caboodle. This means you must run Apache under this user, or use the suexec feature of Apache to switch users. This means you will use the `User` global directive, or the `SuexecUserGroup` directive in the virtual host.

Refer to Apache HTTPD Server documentation online for additional information on configuring virtual servers using SSL, CGI scripts, and suexec.

9.4 Webster

The Webster NodeBrain Module may be used as the web server hosting the NodeBrain Planner. Be careful when using this option, because it introduces some remote control features only appropriate for administrators of the caboodle. If you want to make the Planner available to users you don't want to also have administrative rights to the caboodle and the application account, you should use the Apache option for those users.

To use this option, create a file called '`AccessList.conf`' in the '`security`' subdirectory of your caboodle. The file needs only one line starting with "a," followed by the common name (CN) in your browser certificate, followed by a semi-colon, and optionally a comment if the CN is not self documenting. You may add additional lines if you want to have multiple administrators.

```
a,CN; # comment
```

Next, enable the webster node in the Caboodle agent. This is done using the `nbkit edit` command.

```
$ nbkit caboodle edit Caboodle
```

You will be dropped into `vi` and the bottom of the file will look like the following block. A disabled block starts with `"]!"`. To enable the block, remove the `"!"`. If port 49443 is used on your server, you can change to an unused port.

```
]!
 Code{
 # Webster - Tiny Web Server
 define webster node webster;
 webster. define uri cell "https://0.0.0.0:49443";
 webster. define Config cell "etc/CaboodleConfig.conf";
 webster. define Filter cell "plan/CaboodleFilter/CaboodleFilter.nb";
}
```

When you `:wq` out of `vi`, respond `y` when asked if you want to compile the plan. Your Caboodle agent should automatically start, or restart if it was already running.

Now just point your browser to https://*hostname*:49443/planner/Planner.cgi, using your alternate port if you changed it. If you have more than one certificate installed in your browser, select the one that will be trusted by your server. This means, the one with the common name (CN) specified in your '`security/AccessList.conf`' file and signed by the private key of a trusted certificate in your '`security/TrustedCertificates.pem`' file.

The additional functionality provided by this option may be accessed via the [Webster] item in the upper right menu bar. Use the upper left menu bar for [Bookmarks], [Directory], and [Command] functions. Return to the Planner by selecting [Planner] in the upper right menu bar.

The [Command] option enables remote execution of NodeBrain commands in your Caboodle agent. However, a firewall restricts the commands you can issue via this interface. You manage this firewall using the CaboodleFilter plan.

10 Plan Formats

A plan is normalled stored as an XML document, but may also be represented temporarily in a line editor format, or Planner web display format. This chapter introduces these formats. In later editions, more detail will be provided. For now, the formats are illustrated by example.

10.1 XML Document Format

The XML schema for a plan document is relatively simple as shown below using the CaboodleMailAdmin plan as an example.

```
<?xml version='1.0'?>
<plan scheme='_Subscription' schema='table' version='0.8'>
<description name='CaboodleMailAdmin' title='Administrator Alarm Subscription'>
This plan specifies alarm subscription conditions.
</description>
<guide>
</guide>
<table>
<attributes>
<attribute id='1' time='1362271999' user='admin' name='address'
 title='Email address of subscriber' type='text' for='opt'>root@localhost </attribute>
<attribute id='2' time='1362271999' user='admin' name='method'
 title='Type of alarm adapter' for='opt'>_Mail</attribute>
<attribute id='3' time='1362271999' user='admin' name='form'
 title='Adapter to format alarm content' for='opt'>_Mail</attribute>
<attribute id='4' time='1362271999' user='admin' name='term'
 title='Name associated with condition'/>
<attribute id='5' time='1362271999' user='admin' name='expression'
 title='NodeBrain cell expression based on alarm attributes' type='cell'/>
</attributes>
<rows>
<row id='1' time='1362271999' user='admin'><cell id='4'>all</cell><cell id='5'>1</cell></row>
</rows>
</table>
<history>
<change id='1' time='1362271999' user='admin' update='Full plan update' reason=''/>
</history>
</plan>
```

10.2 Line Editor Format

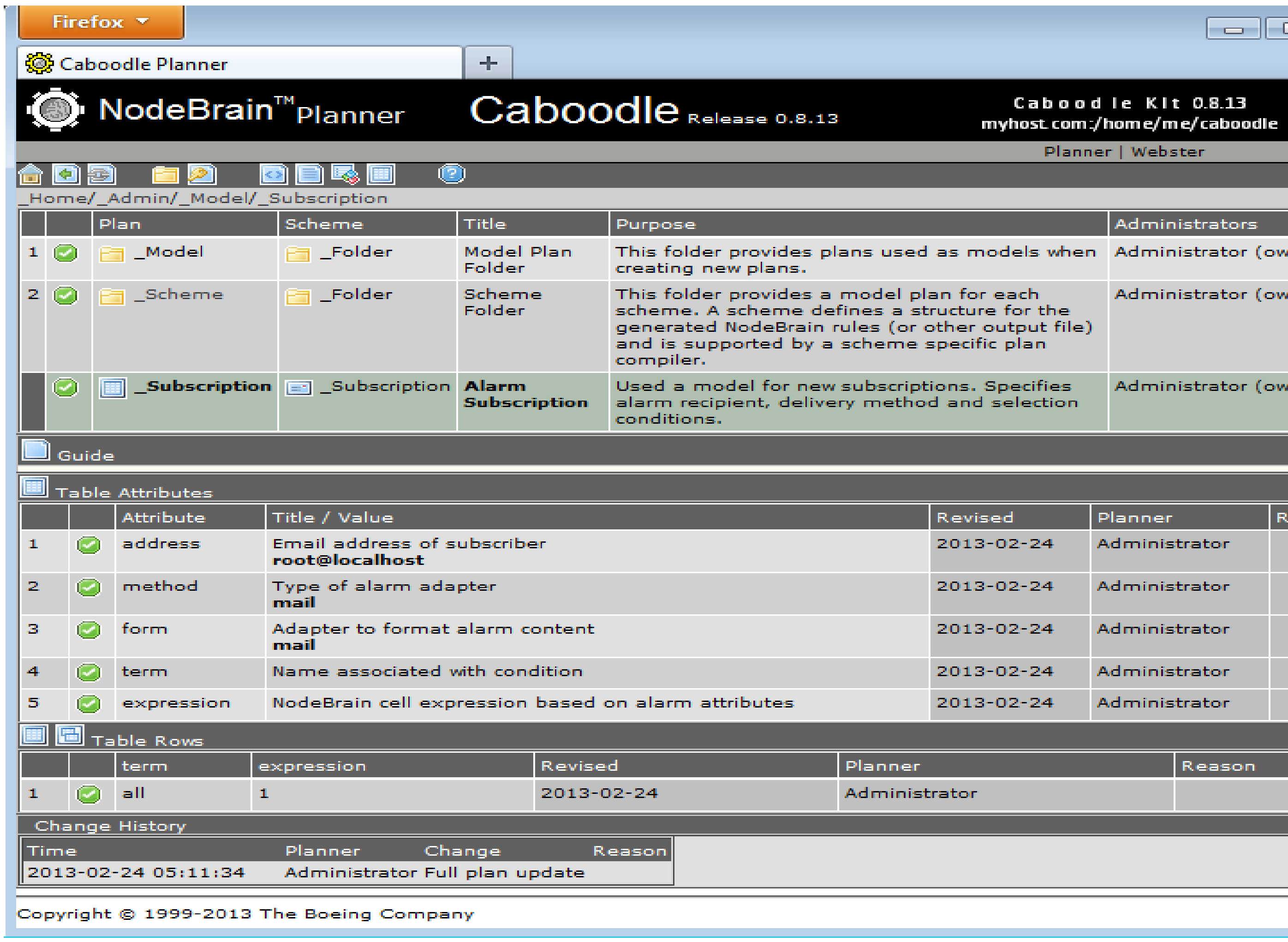
The line editor format is designed to be easy to edit with a line editor. Column one of each line is special in this format. Compare the following representation of the Caboodle-MailAdmin plan to the XML representation above. Change history information is not currently included in this format, and will be dropped when a plan is converted back to XML document format.

```
.scheme: _Subscription

.title: Administrator Alarm Subscription

.purpose{
 This plan specifies alarm subscription conditions.
}

.guide{
}

)table

.address: root@localhost
 title: Email address of subscriber
 type: text
 for: opt

.method: _Mail
 title: Type of alarm adapter
 for: opt

.form: _Mail
 title: Adapter to format alarm content
 for: opt

.term
 title: Name associated with condition

.expression
 title: NodeBrain cell expression based on alarm attributes
 type: cell

>
 term:
 expression:
]
 term: all
 expression: 1
```

10.3 HTML Document Format

Plans are converted from XML to HTML format by the Planner. For authorized users, additional HTML elements are included to enable plan editing. For more information on this format, consult the online help provided by the Planner.

The HTML format uses icons created by Mark James at http://www.famfamfam.com.



Firefox

Caboodle Planner

NodeBrain™ Planner Caboodle Release 0.8.13 Caboodle Kit 0.8.13 myhost.com:/home/me/caboodle

Planner | Webster

_Home/_Admin/_Model/_Subscription

		Plan	Scheme	Title	Purpose	Administrators
1	✓	_Model	_Folder	Model Plan Folder	This folder provides plans used as models when creating new plans.	Administrator (ow
2	✓	_Scheme	_Folder	Scheme Folder	This folder provides a model plan for each scheme. A scheme defines a structure for the generated NodeBrain rules (or other output file) and is supported by a scheme specific plan compiler.	Administrator (ow
	✓	**_Subscription**	_Subscription	**Alarm Subscription**	Used a model for new subscriptions. Specifies alarm recipient, delivery method and selection conditions.	Administrator (ow

Guide

Table Attributes

		Attribute	Title / Value	Revised	Planner	R
1	✓	address	Email address of subscriber **root@localhost**	2013-02-24	Administrator	
2	✓	method	Type of alarm adapter **mail**	2013-02-24	Administrator	
3	✓	form	Adapter to format alarm content **mail**	2013-02-24	Administrator	
4	✓	term	Name associated with condition	2013-02-24	Administrator	
5	✓	expression	NodeBrain cell expression based on alarm attributes	2013-02-24	Administrator	

Table Rows

		term	expression	Revised	Planner	Reason
1	✓	all	1	2013-02-24	Administrator	

Change History

Time	Planner	Change	Reason
2013-02-24 05:11:34	Administrator	Full plan update	

Copyright © 1999-2013 The Boeing Company

Index