Installing and Configuring Big Sister

Thomas Aeby

27th January 2003

Abstract

Many Big Sister users keep complaining about the bad documentation. Unfortunately they are right. This manual is a trial for making things better and is intended for accompanying Big Sister users during their first steps getting it up and running.

Contents

Chapter 1

Installation

1.1 Pre-requisits

The first step in getting a working installation of Big Sister is to download the source package or one of the binary release packages from

```
http://bigsister.graeff.com/download/
```

At the time this manual was written there was a choice between

- the source package
- Big Sister pre-installed for Windows systems (zip archive containing non-compiled but otherwise ready-to-use Big Sister installation, ActivePerl interpreter needed)
- Big Sister binary for Windows (zip archive containing compiled Big Sister executable without a Perl interpreter)

By the time you read this manual other packages - namely RPMs for Linux systems - may be available. Please check the download page and select what suits your needs best.

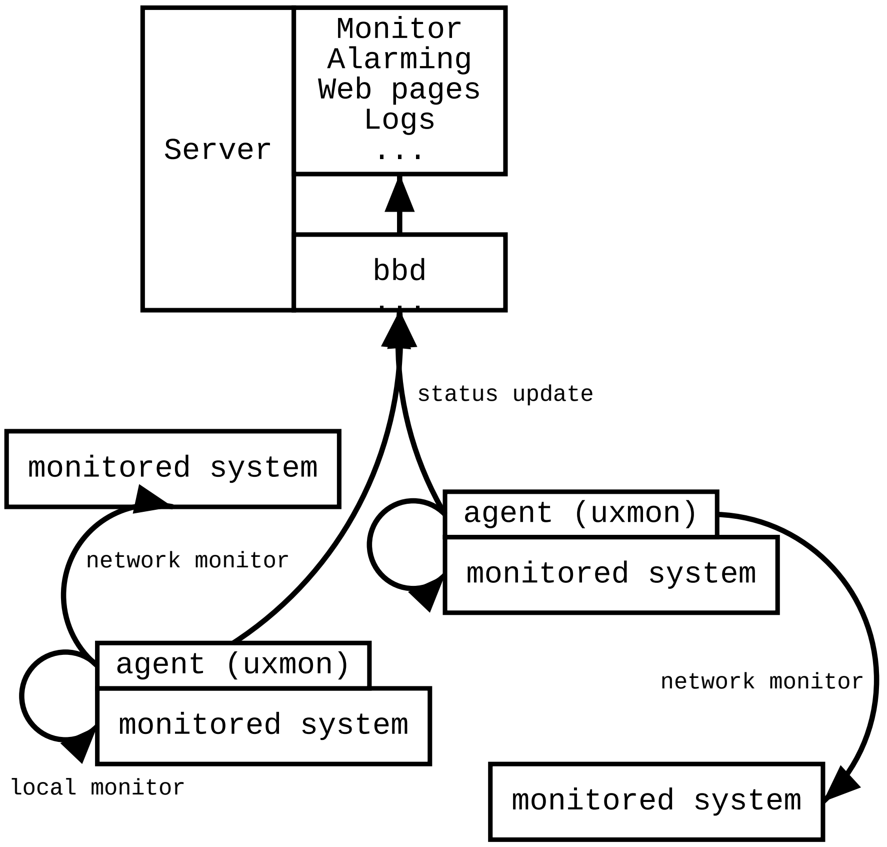

Figure 1.1: Big Sister Components

Depending on what package you downloaded and what features you want to use you will need additional software, e.g. a Perl interpreter, specific Perl modules, etc. All these extra-software is listed in the chapter describing the installation process in your specific case.

1.2 Big Sister Components

Big Sister is composed of a few components usually being hosted on different systems (see figure 1.1). An ordinary Big Sister environment will contain one Server running the Big Sister Server (`bbd`) and the Big Sister Monitor (`bsmon`). `Bbd` is responsible for the communication with the agents including checks if agents are permitted to do certain operations. `Bbd` then passes all the information it gets from the agents to the `bsmon` as is. `Bsmon` processes this information and builds status pages, does alarming, etc. Complex environments may contain multiple interlinked servers.

Every monitored system or service is monitored by a Big Sister Agent (`uxmon`). Some tests the agent performs are only applicable to the system hosting the agent while others work via the network. Usually you will install agents on all the systems you want to monitor provided there is an agent implementation working on the respective system available. You can monitor systems like switches, routers, etc. not running an agent via network (e.g. via SNMP). Only a limitted set of checks works via network though.

It is possible to use the Big Brother agent as a replacement for the Big Sister agent (e.g. for Netware). However, some features of Big Sister (e.g. performance data collection) will not be available in this case.

1.3 Paths

Big Sister puts its binaries and data in specific directories. Depending on your installation the paths pointing to this directories will differ. If you build Big Sister from source you are free to move most of these directories to the location you prefer. The Windows packages allow you to relocate the whole Big Sister package (the `Root` directory) but not each of the directories with special meanings.

What	**Default**	**Windows**	**RPM**
Root	`/usr/local/...` `.../lib/bs`	`C:\bigsis`	`/usr/lib/bigsis`
Binaries	`{Root}/bin`	`{Root}\bin`	`{Root}/bin`
Agent Bin.	`{Root}/uxmon`	`{Root}\bin`	`{Root}/uxmon`
Web Pages	`{Root}/www`	`{Root}\www`	`/var/lib/...` `.../bigsis/www`
RW-Files	`{Root}/var`	`{Root}\var`	`/var/lib/...` `bigsis/var`
local conf.	`{Root}/adm`	`{Root}\adm`	`{Root}/adm`
shared conf.	`{Root}/etc`	`{Root}\etc`	`{Root}/etc`
CGI-URL	`/cgi/`	`/cgi/`	`/cgi/`

The `CGI-URL` is not a physical path - it is rather the path where Big Sister expects its CGI programs to appear when accessed from outside world via the web server.

1.4 Installing Big Sister

1.4.1 Installation from source

When installing Big Sister from the source package you will need at least a working copy of the `make` utility, the Bourne shell, and a working Perl interpreter. This means that you will probably not be able to install Big Sister from source directly on a Windows machine and this section therefore just assumes you are installing on a Unix box.

Before you can proceed you should decide where to store your various Big Sister files (see section 1.3). If this is your first installation it might be a good idea to just use the defaults. You should consider a few points:

- The Web-Pages directory must be accessible via a web browser, probably you will run a web server for this purpose
- Other directories than the Web-Pages directory should not be accessible via a web server for security reasons
- Big Sister will excessively write to the Web-Pages and var directories. They should be located on a fast disk.
- Config files are split into two directories: the adm- and etc- directory. The idea behind is to keep files shared (e.g. via software distribution) between different installations in etc and the files local to the respective machine in the adm directory.

Usually you will install Big Sister under its own user account (default: `bs`) for security reasons. Big Sister daemons will run under this account. Before you can install Big Sister you have to create this account.

You are still with us? Well, then we can start installation. Provided you have already unpacked your Big Sister package, you are logged in as root and you changed directory into your package directory a simple

```
make install
```

should do the trick. However, this will use the defaults for all the installation options. To install with non-default options you execute something like

```
make install OPTION1=arg1 OPTION2=arg2 ...
```

Valid options are:

DEST - the root directory (defaults to `/usr/local/lib/bs`)

EXEC - the path where the root directory can be found when Big Sister is running (useful e.g. when installing in a first directory, let's say /tmp/bigsister on your test system, then copy all the files over to /usr/local/lib/bs of your production system, in this case you would choose `DEST=/tmp/bigsister EXEC=/usr/local/lib/bs`). This defaults to the same as the `DEST` option and setting this will only be useful in rare cases.

USER - The name of the user Big Sister will be installed (defaults to `bs`)

CGIPATH - Big Sister comes with a few CGI programs to be run by the web server. `CGIPATH` is the root URL of the CGI directory where these programs will be stored.

WEBROOT - The directory `{DEST}/www` will be the location where Big Sister stores its web pages. This directory must be accessible via a web browser (usually via a web server). The `WEBROOT` option must contain the URL under which this directory is accessible. This defaults to `/`.

PERL - The path to your perl interpreter. By default `make` will look it up in your program path. You will have to use the `PERL` option if the `perl` command is not in your path.

INCLUDE - the root path under which Big Sister will find its own perl modules. This defaults to the same as `DEST` and I cannot imagine a situation where you would choose something different.

So if you would like to install Big Sister in `/usr/lib/bigsis`, run it under the account `bigsis`, you have configured your web server to display Big Sister pages under `http://myhost/bigsis/` and make CGIs accessible via `http://myhost/bigsis/cgi` then the command you invoke is

```
    make install DEST=/usr/lib/bigsis USER=bigsis \
         WEBROOT=/bigsis CGIPATH=/bigsis/cgi
```

1.5 Installing Windows binary

Before you can install Big Sister on a Windows system you need the utilities `InstSrv.exe` and `SrvAny.exe`. These two utilities are part of the NT Resource Kit. Microsoft's license restricts us from distributing the tools with Big Sister. Anyway, if you do not have a copy of the Resource Kit you will probably be able to find them via Altavista (search for "inststrv.exe" on `http://www.altavista.com/`).

Once you successfully located your copy of `InstSrv` and `SrvAny` you can place them somewhere in the path, e.g. in your `system32` directory in your Windows directory.

You are now ready for unpacking the zip archive. Place the resulting `bigsister` directory wherever you like and then double click the `bin/install32.exe` executable. The tasks performed by install32 are:

- register Big Sister in the registry (install path)
- register Big Sister services in the Service Manager

Now start the Service Manager and decide which services your Windows box should start on boot. Three Big Sister services are registered: The agent `uxmon`, the display server `bbd` and the monitor `bsmon`. The server must run at least `bbd` and `bsmon`, while on agent systems only the agent is needed. Do not be irritated by the strange names listed in the service manager - during the next reboot Windows will adjust them.

1.6 Post-installation tasks

For various reasons you might have to change some of the configurations the installation procedure made - e.g. because you have installed from a binary package. Also it might be necessary to install additional software to enable some Big Sister features.

1.6.1 Installing Perl modules

Note: This section only applies if you have installed Perl on your machines running Big Sister. If you are using the binary Windows package you already got all the necessary Perl modules implicitly.

The following Perl modules are necessary to enable certain functionality:

SNMP - All the SNMP functionality in the agent and the server bases on Simon Leinen's SNMP module available from

`http://www.switch.ch/misc/leinen/snmp/perl/`

GD - Big Sister can present system status overviews as a graphical image map. This functionality bases on the GD module available from CPAN (install it on the server system).

Net::SMTP - Alarming is usually done via E-Mail and the `sendmail` program. If you prefer your Big Sister server to directly transmit alarm mails via SMTP - e.g. because you are running Big Sister on a non-Unix system - you will have to install the Net::SMTP module available from CPAN on your server system.

LWP::UserAgent - Web server checking can be done via a simple TCP monitor (`http` test) or via a HTTP-aware monitor (`realhttp`) providing some additional information on the tested web server. In the latter case you will have to install the LWP::UserAgent module on the agent system(s).

You will find a list of CPAN mirrors on

`http://www.cpan.org/`

All of the above modules - except for the GD module - are easy to install. Just follow the instructions in the respective module.

Note: For Linux systems many modules are available via RPM packages from your distributor. ActiveState Perl (Windows) comes with its own easy way of installing modules (see `http://www.activestate.com/`).

1.6.2 Installing RRDTool

Big Sister is able to collect performance data, store it in a database and provide you with nice trend graphics. Currently only one database exactly designed for such purposes is supported: Tobi Oetiker's RRDTool. For this feature to work you will have to download and install RRDTool. At the time this manual was written it was not necessary to install the RRDTool perl modules - having the `rrdtool` command installed in your path was enough. Anyway it is a good idea to install these modules. In near future they might improve performance of your server.

RRDTool is available from

`http://ee-staff.ethz.ch/~oetiker/webtools/rrdtool/`

1.6.3 Installing CGIs

Big Sister's CGI scripts are installed in its `bin`-directory. For security reasons it is not a good idea to just configure your web server to use this directory as a CGI-directory. It is better to either copy the relevant programs over or place symbolic links to them into your CGI directory.

The CGI scripts are:

- bshistory
- bswebalarm

- bswebadmin
- bsgraph

Moving CGIs after installation

During installation you have to point the installation routine to the URL your CGIs will be accessible through. If you installed a binary package you even have no choice at all and Big Sister assumes CGIs are accessible via `/cgi`.

If this does not correspond with your web server configuration any more you will have to edit the files `...www/skins/default/*_PATH.inc` where Big Sister stores some paths. In order to activate your changes you then remove the file `...www/skins/default/cache`.

1.6.4 Web Server

The Big Sister installation routine will not touch your web server configuration in any way. It relies on your web server to be configured in order to:

- make the documents in the `.../www` directory
- execute the CGI scripts (see 1.6.3) on demand

Chapter 2

Configuration

2.1 First steps

Are you curious? Just want to start it up and see what's going on? Ok, let's go. There is one thing you have to do first: On all your agent systems you will have to edit the `adm/uxmon-net` file locate the line

```
    localhost bsdisplay
```

and replace `localhost` by the name of the system hosting your Big Sister server. After that you can start the server by issueing

```
    bin/bb_start start
```

(Unix) or start the `Big Sister Server` and `Big Sister Monitor` services on Windows systems. The server should immediately start listening to the agent/server port (TCP 1984) and create the initial web pages in the `www` directory. Try starting a Web browser and point it to the newly generated `index.html` file.

Now you can proceed to bringing up agents. On Windows systems start the `Big Sister Agent` service via the `Control Panel` / `Service Manager`, on Unix systems remove the file `adm/bb-display.cfg` and `etc/bsmon.cfg` (only keep these files on the server) and use the `bb_start` command (see above) for starting the Agent. You already have done the modification of `adm/uxmon-net`, haven't you?

Now go back to your web browser and reload the viewed pages. If you are using the `bb-display.cfg` initially installed on the server the agents should automatically appear on the status page (this may take one or two minutes).

Try the links `Admin`, `Alarms` and `History` to see if you got the CGI configuration of your web server right.

2.2 Basic Configuration

2.2.1 Daemon startup

Under Windows use the `Control Panel` / `Service Manager` to tell your system which Big Sister services it should startup on boot. See section 1.2 for determining what services should run on which systems.

Under Unix the startup of Big Sister daemons is a little different. There is a shell script named `bin/bb_start` meant to work like a System V init script, thus running

```
    bin/bb_start start
```

will start up the Big Sister daemons while

```
    bin/bb_start stop
```

will shut them down and

```
    bin/bb_start restart
```

will restart Big Sister.

Therefore you can put a copy or a symbolic link to `bb_start` into your init directory (usually one of `/etc/init.d`, `/etc/rc.d/init.d`, `/sbin/init.d`) and create the necessary links in your `rc*.d` directories. If you are running Big Sister on RedHat Linux you can use the `chkconfig` command to enable / disable Big Sister.

`bb_start` will look at the config files in the Big Sister directory for deciding which daemons it will start up. If there is one (or more) `adm/uxmon-net` files present the agent will be started, if there is a file called `adm/uxmon-asroot` the agent will be started under the `root` account (do this only if you know what you are doing!), if a file called `adm/bb-display.cfg` is present `bbd` will start up and finally if `etc/bsmon.cfg` exists `bsmon` starts up. So if you want to use your Big Sister instance as an agent only just remove `adm/bb-display.cfg` and `etc/bsmon.cfg`.

2.2.2 Agent configuration

The agent (`uxmon`) will read its configuration from the file `adm/uxmon-net`. Under Unix multiple `uxmon-net` files may exist – their name must start with `adm/uxmon-net` followed by an arbitrary suffix (note that e.g. `adm/uxmon-net.bak` **is** a valid agent configuration file!). `Bb_start` will in this case start up an instance of `uxmon` for each of these configuration files.

The syntax of the `uxmon-net` file is kept very simple: each entry starts with the name or IP address of a to-be-monitored system followed by a list of checks that should be applied to this system, e.g.:

```
localhost type=ext2 diskfree
```

will run the `diskfree` test against all mounted partitions holding an ext2 file system of the local system. As you can see most of the checks accept arguments. Arguments are always **preceeding** the check and are of the form

```
argument1=value1 argument2=value2 ... check
```

The argument list only applies to the check immediately following the last argument in the list, thus

```
myhost proto=icmp ping ping
```

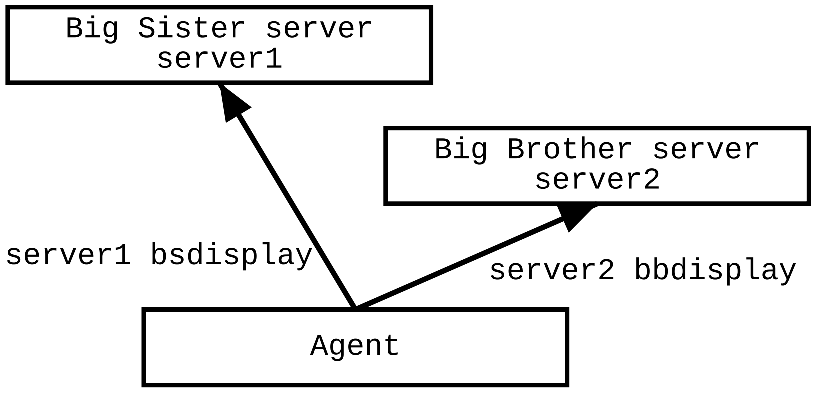

Figure 2.1: Agent to Server

will run two `ping` checks against the host `myhost`, the first one will do an ICMP ping, while the second will do a ping using the default protocol (usually UDP). The `proto` argument does not influence the second `ping` check, but of course you can do something (rather senseless) like

```
    myhost proto=icmp ping proto=icmp ping
```

You will find a complete and hopefully up to date list of available checks with their arguments in the `CONFIG` documentation coming with your Big Sister package.

`uxmon-net` entries may span multiple lines. Usually a line end will automatically end the respective configuration entry. However if a line ends with a “`\`” character the following line is assumed to be part of the entry.

Pointing the agent to your server

Two special pseudo-checks in the `uxmon-net` file point your agent to the server(s) status reports should be sent to: the `bsdisplay` and `bbdisplay` checks. The line

```
    myserver bsdisplay
```

for instance will force the agent to send status information to the server `myserver` where `myserver` talks the Big Sister protocol. You can use `uxmon` in conjunction with a Big Brother server by changing the above line into

```
    myserver bbdisplay
```

In this case the agent will suppress any non-Big-Brother feature and use the Big Brother protocol to talk to the server (see figure 2.1.

Of course multiple `bsdisplay` / `bbdisplay` lines may appear in `uxmon-net`. In this case the agent will report its status information multiple times to (potentially) different servers.

As one would expect the `bsdisplay` pseudo-check accepts a few arguments, e.g. the line

```
    myserver fqdn=no bsdisplay
```

will report status information to `myserver` by stripping domains from all the host names.

Other useful arguments are relevant if you want Big Sister to keep statistics on performance data and are listed in section **??**.

Host aliases

Sometimes it is necessary to differ between the host you are running a check against and the host name reported to the server. For instance imagine you are running a check against a multihomed host (a host with multiple IP addresses) and you want to access this target system via a well-defined network interface. In this case you can use a configuration line like the following:

```
    192.168.1.17(myhost) ping
```

This will run the ping test against `192.168.1.17`. The result of the test however is then reported to be related to the host `myhost` (see figure 2.2).

A host definition of the form

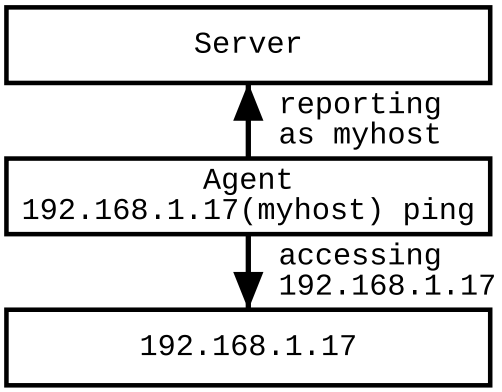

Figure 2.2: Host Aliases

```
name1(name2)
```

is always treated by `uxmon` in the following way: `name1` is used internally by the checks while `name2` is the name reported to the Big Sister server. The server will be completely unaware of the `uxmon` internal name (`name1`).

Check frequencies

By default uxmon runs checks and reports information in 5 minutes intervals. Anyway, some checks might put some load on the target system, or are of no short-term relevance and you prefer to run them less frequently. Other checks might be of extreme importance and should be run more often. For such occasions you can define your own check frequencies. Every check (including the `bsdisplay` and `bbdisplay` pseudo-checks) accept the special argument `frequency`. E.g. the `uxmon-net` line

```
localhost frequency=180 metastat
importantmachina frequency=1 ping
```

will run the metastat check against the local machine every 180 minutes while the ping check is run every minute. Note that the argument name `frequency` is a little bit misleading – it is not really a frequency but rather the time interval in minutes between two runs of a check.

When defining check frequencies keep some rules in mind:

- Running a check more often than the fastest `bsdisplay` pseudo-check is senseless. Check results will only be reported during `bsdisplay` runs, **not** necessarily immediately after each check's run. So the above example only makes sense if you have got for instance the following `uxmon-net`:

  ```
  localhost frequency=180 metastat
  importantmachina frequency=1 ping
  myserver frequency=1 bsdisplay
  ```

- You must run `bsdisplay` checks at least every 10 minutes. The server relies on the agents to report their status rather frequently. If no status information is coming in for 15 minutes the server will assume the agent or communication to the agent is dead and will set status to `purple` (no report).
- The above rule does not apply to other checks. Even if you run certain checks only every few hours the `bsdisplay` check will report the result of the last check run. So the server will not change status to `purple`.

Defaults

Proceeding to more complex `uxmon-net` files you will probably get bored by repeatedly list the same check arguments again and again. Fortunately `uxmon` supports setting defaults for certain arguments. For instance the configuration:

```
localhost frequency=2 type=ext2 diskfree
localhost frequency=2 memory
localhost frequency=2 procs=sendmail procs
fileserver frequency=5 nfs
myserver frequency=2 bsdisplay
```

can be simplified to

```
DEFAULT frequency=2 ALL
DEFAULT type=ext2 diskfree

localhost diskfree
localhost memory procs=sendmail procs
fileserver frequency=5 nfs
myserver bsdisplay
```

the `DEFAULT` statements in this example do

- set the default check interval for all (`ALL`) the checks to 2 minutes
- set the default type for all `diskfree` checks to `ext2`

Of course you can override defaults by just listing an argument with a non-default value as usual. For instance in the example above the interval for the `nfs` check is explicitly set to 5 minutes overriding the default interval of 2 minutes.

Including files

As in every other configuration file you can include the contents of another file using the include statement:

```
include uxmon.include
```

This line will cause `uxmon` to search the `adm`, `etc` and Big Sister root directory (in this order) for a file called `uxmon.include` and – if found – will insert the contents of this file in `uxmon-net`. Of course you can use absolute or relative paths with the file name argument. Note that relative paths also will be searched in `adm`, `etc` and the root directory. Therefore if you e.g. use the statement

```
include etc/uxmon.include
```

this will normally force `uxmon` to include the file `uxmon.include` in the `etc` directory. However, if e.g. a file `adm/etc/uxmon.include` exists, this one is included in preference to `etc/uxmon.include`!

The file name argument of an include statement may contain variable references like in

```
include uxmon.$HOST
```

Here `$HOST` will be replaced by the hostname of the system hosting `uxmon`. Currently only `$HOST` contains a defined value. Environment variables are **not** visible to the `include` statement.

2.2.3 Display server configuration

The main server configuration file is `adm/bb-display.cfg`. With very few exceptions every display related configuration appears somewhere in this file. The suggested structure of the file is:

- Options
- System names and groups
- Web pages

Each of these sections is composed of a number of **statements**. Statements always start on a new line and are prefixed by a "`%`" character. Most statements take arguments, so a valid statement looks like e.g.:

```
%Autojoin new NEW
```

(Autojoin with first argument `new` and second argument `NEW`)

See the section on `bb-display.cfg` in the CONFIG documentation for a complete list of known statements. Note that the above structure is mandatory.

The options section

In the options section you specify various not necessarily display related parameters. Statements belonging to the options section include

Option – specify (boolean) options

Autoconn – switch on the autoconn feature for specific hosts

Pager – set the pager program used by Big Brother agents

A valid options section might look like

```
# Set some options:
#  - do not forget grouping information on restart
#  - only update files in www/html when status
#    text or status color changes
#  - no logging in www/logs
%Option +KeepGroups -ImmediateHTML -BBLog

# Turn autoconn feature on for host myagent
%Autoconn myagent

# Pager program for Big Brother clients is
# BBs bb-page.sh
%Pager /usr/local/lib/bb/bin/bb-page.sh
```

Using the autoconn feature The `%Autoconn` statement indicates to `bbd` (**not** `bsmon`!) for which hosts it should enable the autoconn feature. Whenever an agent connects to `bbd` for transmission of status information `bbd` looks up the system hosting the agent in the list of autoconn hosts. If it is listed there `bbd` automatically generates a `green` status message for this hosts `conn` (connection) status. When the agent does not connect to the server for more than 15 minutes the status is automatically changed to `red`.

The system names and groups section

Big Sister uses groups at various places. You will meet them when defining what systems' status will appear on which web page (see section 2.2.3) as well as when setting up your alarming rules (see section 2.2.4).

In the system names and groups section of `bb-display.cfg` you specify displayed system and group names and define the group hierarchy. The most important statement you will want to remember is the `Groups` statement. All the lines between a `Group` statement and the next valid statement will be treated as group and/or name specifications. Each line is of the form

```
    hostname(Displayed Host Name) GROUP1 ... GROUPN
```

or

```
    groupname(Displayed Group Name) GROUP1 ... GROUPN
```

where hostname/groupname is the name of a host (group resp.) and the string in parenthesis is the description of the host or group shown on web pages. `GROUP1` through `GROUPN` are names of groups hostname/groupname is a member of. Groups are created when they are first referenced so you can use an arbitrary name for both the `groupname` or `GROUPN` arguments. Groups may themselves be members of other groups. It is a good idea to avoid circular groups though.

Time for an example: Assuming `washington`, `paris` and `london` are three systems monitored by Big Sister the grouping statement

```
    %Groups
    washington(Our server in Washington) USA SERVER
    paris(Our server in Paris) EUROPE SERVER
    london(Our server in London) EUROPE SERVER

    EUROPE(Good old Europe) WORLD
    USA(The United States) WORLD

    WORLD(all the continents) UNIVERSE
    SERVER(all our servers) UNIVERSE
```

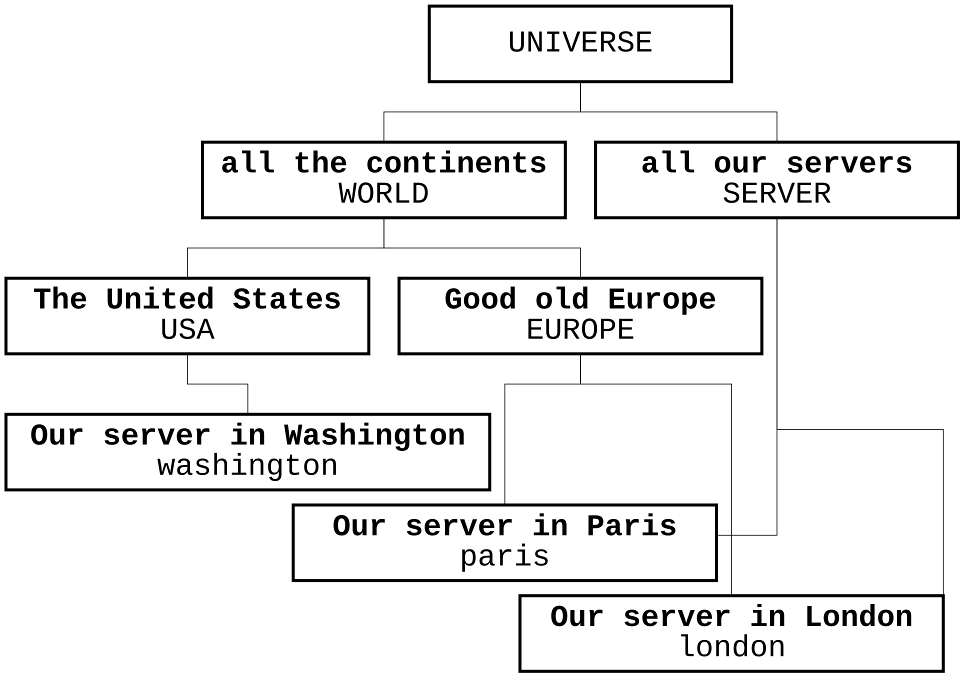

Figure 2.3: Group hierarchy example

will create a group called `USA` containing only the system `washington`, the group `EUROPE` containing the systems `paris` and `london`, the group `WORLD` containing the groups `EUROPE` and `USA`, the group `SERVER` containing all three systems, and finally the group `UNIVERSE` containing the groups `SERVER` and `WORLD` (see figure 2.3).

Automatically joined groups Big Sister can automatically add a host to special groups at the time the first status message for the host comes in. Via the `Autojoin` statement you declare which groups Big Sister will use for this purpose. Basically there are three available autojoin features:

- `%Autojoin new GROUPNAME` – hosts joining in that are not member of any group yet (semantics: "they are new" / "not defined yet") will join the group named `GROUPNAME`. This is especially useful for detecting hosts which are already monitored by an agent but which are not correctly configured on server side.
- `%Autojoin all_hosts GROUPNAME` – any host (but not groups!) becomes member of the group named `GROUPNAME`.
- `%Autojoin all GROUPNAME` – any host or group becomes member of the group named `GROUPNAME`.

The web pages section

The web pages section defines what web pages are generated as well as what they look like. A few statements (like the `skin` and `Logskin` statements) are of a global nature while others (like `title`, `table`, etc.) are only applicable in conjunction with a `Page` statement.

Mainly you will define the basic look of your web pages via the `skin` and `Logskin` statements, then you describe the pages you would like to be generated and their contents via the `Page` statement and its "children" (`title`, `refto`, `table`, `ref`, `itemref`, `image`). Each page description starts with a `Page` statement as e.g.

```
    %Page top The_main_page
```

(create a web page called `top.html` with title “The main page”) Note that the title must not contain spaces. To bypass this limitation underscores will be replaced by spaces.

A single `Page` statement will not create much more than an empty page. The lines immediately following it should define some contents. Most important in this respect is the `table` statement. Followed by one or more groups it will insert one table per group containing the hosts or groups in the respective group and their status. So e.g. (the group names and members are taken from the example in section 2.2.3)

```
%Page top The_main_page
%table EUROPE USA
```

will create a page with filename `top.html` containing two tables, one containing all the members of the group `EUROPE` (namely `paris` and `london`), the other containing all the members of the group USA.

Note that `table` will only list the members of a group at the next hierarchy level, so e.g.

```
%table WORLD
```

will list `EUROPE` and `USA`, **not** `washington`, `london`, `paris`. You can control how deep `table` digs by prefixing the group name(s) with a “+” sign as in

```
%table +WORLD
```

This will list the group members two levels below `WORLD` (`washington`, `london`, `paris`). Multiple “+” signs will make `table` go deeper in the hierarchy.

All the other statements (except for the `image` statement documented in section 2.3.3) will not create contents themselves. Instead they modify the behaviour of the following `table` statement.

- `%title Any_title` – specifies the tables title. if you use the special word `none` instead of a title tables are created without any title, the special word `auto` makes `table` use the display name (see section 2.2.3) of the groups the respective tables are created from. The latter is the suggested variation.

- `%itemref directory` – if `directory` does not equal the special word `none` all the status lights in the following tables will refer to the file `host.check.html` in the sub directory `directory` of the www directory. So by clicking on a status light the browser will open the respective file. Commonly used `itemref` statements are

  ```
  %itemref html
  ```

  (point to where Big Sister stores its html-ized status messages) or

  ```
  %itemref logs
  ```

  (point to where Big Sister stores its ASCII text version status messages). The latter only works if the `BBLog` option is enabled (see section 2.2.3).

- `%refto url` – whenever host names or group names appear in a table Big Sister will create a hypertext link pointing to `url#hostname`. Usually `url` will be the name of a page created via `bb-display.cfg` and is used to point from tables showing consolidated status information to more detailed tables. Big Sister will automatically create the necessary labels each time it inserts a table in a web page. Of course you are free to use URLs pointing somewhere outside the set of auto generated pages. Note that `refto` always applies to all rows of a table. Lets make an example:

  ```
  %Page top Top_Page
  %refto detailed
  %table WORLD

  %Page detailed Host_Details
  %itemref html
  %refto http://oursite.com/serverlist.html
  %table EUROPE USA
  ```

  This will create two web pages `top.html` and `detailed.html`. On the first page only a single table containing rows for “Good old Europe” and “The United States” is shown. These two labels are linked to the second page where there are two tables, one listing all the hosts in the group `EUROPE`, one listing the hosts in the group `USA`. The labels in this two tables are linked to a manually created descriptive page `...serverlist.html`.

`refto` accepts a few special pseudo-URLs: `refto none` will suppress hypertext linking, `refto self` will create links pointing to the same page.

- `refto name url` – the `refto` statement explained above always takes effect on a whole table. However sometimes it is necessary to link individual hosts or group to individual pages. Another flavour of the `refto` statement supports exactly this. Via e.g.

  ```
  %refto washington /servers/washington.html
  ```

  you force the following `table` statements to link each appearing `washington` label to the page /servers/washington.html#washington. No matter which comes first an individual `refto` always overrides a global one, so that e.g.

  ```
  %Page top Top_Page
  %refto EUROPE europe
  %refto USA usa
  %refto all_the_rest
  %table WORLD

  %Page europe Europe
  %refto none
  %table EUROPE

  %Page usa USA
  %refto none
  %table USA
  ```

  will work correctly, thus creating three pages, one being an index page showing consolidated status for Europe and USA pointing to the respective pages listing the servers in Europe and USA.

  Individual `refto` entries are kept across `Page` statements. If you need to limit scope of entries you can use the special pseudo-URL `clear` as in this example:

  ```
  %Page top Top_Page
  %refto EUROPE europe
  %refto USA usa
  %refto servers
  ```

```
%table WORLD
%refto clear
%table WORLD
```

This will create a table with links to the Europe and USA pages, then another table with links to the `servers` page.

Skins

While Big Sister strictly limits the contents of web pages to a few elements (actually tables and image maps) the way web pages look is configurable via the so-called skin mechanism. A skin defines the layout of pages – to give some examples skins decide if tables get borders, if the background is white or yellow, if legends are at the top or at the bottom of a page or not present at all, if status lights are round and blinking or rather triangular and static, and so on.

Most skins do not define the whole palette of possible features. Instead, there is a basic skin (the `default` skin) and various skins modifying only a few layout elements – e.g. the `white_bg` skin will change the background to white, the `static_lamps` skin will replace the blinking status lights by non-blinking lights, etc. A whole set of such skins is called a skin set.

Skins appear in conjunction with the `Logskin` and `skin` statement. `Logskin` is used to specify the skin set Big Sister uses when creating the web pages for each checks detailed status while `skin` sets the default skin for everything else.

Example:

```
%Logskin white_bg
%skin static_lamps,structured_bg,frames
```

This will make Big Sister create the HTML log pages with white background in place of the default one and all the other web pages with static status lights, a textured background and using HTML frames.

Note that the default skin is automatically part of any skin set, there is no need for explicitly listing it.

Some of the available skins are:

title_in_table – make table titles part of the table

white_bg – set pages' background to white in place of the default colored background

structured_bg – another alternate background

static_lamps – display non-blinking status lights

frames – frame optimized skin

bigbro13 – Big Brother 1.3 like look

alt_contentsicons – especially ugly status lights in contents frame

Frames

Have you already read section 2.2.3 concentrating on skins? Pages displayed in frames are in fact just a special way of layouting pages, therefore you need only use a frames-enabled skin set (e.g. one including the skin `frame`) and you get a layout using frames.

Anyway, there is one little problem. It is not sufficient to create all the status pages, additionally an index page defining the frame set as well as the non-status frames are needed. This is the realm of the `Frameset` statement:

```
%Frameset index top Monitored_by_Big_Sister
```

in `bb-display.cfg` will create an index page called `index.html`, the initial page displayed when entering index.html is `top.html` and the title of the frame set is “Monitored by Big Sister”.

The statement will only work if the specified skin is frame-aware. If the skin defines a menu frame only the pages defined up to the `Frameset` statement will be respected.

2.2.4 Configuring alarming

Big Sister implements alarming in a server based manner. The agent is responsible for determining if a system or service is working correctly (“green”), if it is critical (“yellow”) or it has failed (“red”) – other stati do exist but are not relevant to alarming. This status is noticed by the alarming module of the server. Depending on the configuration file `adm/bb_event_generator.cfg` the server generates alarms on status changes.

The alarming configuration mainly consists of a set of rules. Each rule consists of a pattern matched against all status change, a definition of dependencies and a description of the action to be taken when an alarm is raised. The first two elements describe under what circumstances an alarm is to be raised while the last one describes how actually the alarm is raised. Using this simple approach a few things can easily be configured either for individual checks, for individual hosts or for whole groups:

- wait for a defined time period before reporting an alarm and do not report an alarm if the problem goes away within this period
- regularly send reminders telling the administrator that a problem persists until the problem goes away
- do not repeatedly send alarms for a multiply occurring problem
- behave different depending on time of day or day of week (e.g. postpone alarms raised during the night to the early morning)
- suppress alarms depending on what status other systems/services are in (e.g. do not report that a system is unreachable when Big Sister already knows that the whole network the system is connected to is down)

The main disadvantage of the existing rule based alarming configuration is that it is very hard to find a simple way to explain how it works. Unfortunately you will just have to read the whole section and hopefully understand the configuration at the end.

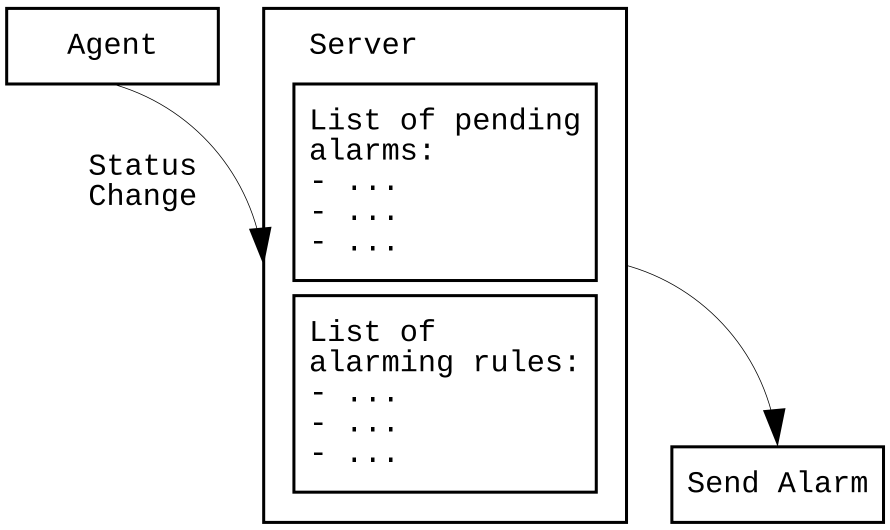

Figure 2.4: Status Changes result in Alarms

Rules

An alarming rule in the `bb_event_generator` file always starts with a pattern followed by a description describing what actions should be taken if the pattern matches. Every time a status change is noticed the alarm generator does two things:

- go through the pending alarms and check if the status change has some effect on one of them
- if the status change is not related with one of the pending alarms: go through the list of rules, select all the matching rules and raise an alarm depending on their descriptive part

Usually each line in the configuration file represents one rule. Of course like in most Big Sister configuration files empty lines and lines starting with a '#' character are treated as comments and are therefore simply ignored. A rule may span multiple lines: Lines terminated with a '\' character are joined with their following line.

Patterns – "when to do things"

The most simple form of a pattern is a `host.check` pattern. A rule

```
    foo.cpu mail=nobody
```

(where `foo.cpu` is the pattern and `mail=nobody` is the description) for instance matches only status changes for the host `foo` and the `cpu` check. The above rule tells Big Sister to forward alarms for `foo.cpu` to the user `nobody`.

Now let's assume you do not want to list each individual system and check in the rule file. The Alarm Generator accepts one single wildcard – `*` – matching any check or any host:

```
    *.cpu mail=nobody
```

extends the above rule to be effective for any `cpu` check of any host while

```
    foo.* mail=nobody
```

matches any status change reported for host `foo` and finally

```
    *.* mail=nobody
```

matches any status change for any host.

Of course you may want to address a group of hosts - haven't you spent hours setting up groups after reading section 2.2.3? Exactly these groups are also visible to the alarm generator. By prefixing a host name with a '`@`' character you point Big Sister to match a group rather than a single host so that a rule like

```
    @USA.* mail=nobody
```

for instance applies to any status change reported for any system being member of the group `USA`.

So far so good. Sometimes it is very useful to be able to make alarming behave different depending on when a status change is detected - maybe you just refuse to be woken up by your beeper during the night or you want get alarms via another medium during working hours. For this purpose the patterns can contain so-called pre-conditions. In the rule

```
@USA.*{weekday Sat,Sun} mail=pikett
```

the stuff in parenthesis is a pre-condition. The rule will only match status changes for any system being member of the group `USA` reported during the weekend. Another useful precondition is the `daytime` condition. This rule

```
*.*{daytime 17:00-07:00} down=never
```

for instance will suppress (`down=never`) any status change reported between 5pm and 7am. Of course conditions can be combined using `and` and `or`, so

```
*.*{daytime 17:00-07:00 or weekday Sat,Sun} down=never
```

will suppress any status change reported between 5pm and 7am or during weekend.

Description - "what to do"

Associated with each pattern there is a description in the form of a bunch of definitions. This set of definitions describes what actually will be done if a status change matching the pattern occurs. The rules will be processed in the order they appear in the configuration file and if multiple patterns match all the definitions will cumulate. Definitions appearing later in the file will overwrite definitions appearing earlier, e.g.:

```
*.* mail=alarm@nowhere.org delay=5
*.cpu delay=100
```

If a status change for `myhost.conn` is reported then only the first pattern will match resulting in a description of:

```
mail=alarm@nowhere.org delay=5
```

while if a status change for `myhost.cpu` is reported both patterns will match and the resulting description would look like:

```
    mail=alarm@nowhere.org delay=100
```

thus the `mail` definition will be taken from the first rule while the `delay` definition of the second matching rule will replace the concurring definition in the first rule.

It is a good idea to place more general rules near the start of the configuration file and more specific rules near the end. E.g. a rule associated with the pattern `*.*` is working like default settings since it will match every single status change. Consider

```
    *.* mail=alarm delay=5 down=yellow up=green prio=5
```

Placed at the very start of the configuration it will initialize the settings for `mail`, `delay`, `down`, `up` and `prio`. Later rules may re-set one of these settings by at the same time inheriting all the other settings.

It is time to specify the meanings of all these settings. As you already discovered settings resemble variable definitions. Now some of these variables have special meanings:

prio a number between 0 (completely unimportant) and 100 (extremely critical) describing the importance of the alarm. The priority settings can be used in pre-conditions (see 2.2.4 and are otherwise passed through to the alarming methods. E.g. for alarms sent via E-Mail the priority will only appear in the message text and does not have any influence on how the alarm is treated.

down is a status color. Status colors equal or below this color are considered a failure, thus an alarm is raised if a status change occurs from a color "above" this color. E.g.

```
    down=yellow
```

will make Big Sister raise an alarm if a status changes from green to yellow or green to red but not if a status changes from green to purple.

up works similiar to `down` but defines when a status will be considered to go up. By default `up` is the same as the next "higher" color of `down`. Sometimes it might be useful to re-define this. Consider

```
    down=yellow up=green
```

This will raise an alarm on status change from e.g. green to red. If the status goes up to purple (aka. no information) the alarm will not be cleared. It will only be cleared as soon as we get a green (aka. everything's ok).

delay is a time in minutes. Whenever an alarm is raised it first goes into a pool of alarms just about to be raised. It stays in this pool for the `delay` time. If during this time the alarm condition clears (service up) the alarm is silently dropped. If an alarm is still pending after the `delay` time the alarm is finally sent to the administrator.

keep is also a time in minutes. After an alarm condition clears the alarm is kept active for the `keep` time period. Only after this time the administrator will get an "alarm cleared" message and the alarm will go in the pool of old alarms. Do not ask me what this is useful for.

norepeat after an alarm is cleared it goes into the pool of old (remembered) alarms and stays there for the `norepeat` time period. As long as an alarm is either pending, active or remembered no new alarm for the same host/status is raised. The meaning of "norepeat" therefore is: Do not send an alarm again for the same condition for this delay. The `norepeat` period starts when the alarm is **raised**. Therefore it is of course possible that the `norepeat` delay is already over when an alarm gets cleared and therefore the respective alarm is immediately thrown out of the pool of remembered alarms.

mail names the recipient (mail address or pager number or whatever depending on the value of the `pager` variable) alarms are sent to.

pager tells Big Sister which program it should use for sending out alarms. The default is "notify" which is a pager program included with Big Sister. It is not a bad idea to just keep this default.

repeat is a time period in minutes. If an alarm stays active for some time every `repeat` minutes Big Sister will send a reminder message to the recipient of the original alarm message. It is suggested to use this feature only for really important alarms since most administrators will probably just get annoyed when continuously reminded of the same failures. **Note**: This is not related to `norepeat` in any way.

repeatprio is the priority (see above) of reminder messages.

trap If `trap` is set Big Sister will sent out an SNMP trap on each alarm raise/clear (see also 2.3.7). The value of `trap` is of the form

```
trap=community@host
```

You will find the Big Sister MIB (if you do not know what a MIB is you do not need one) as well as format file for HP Openview in the `contrib` directory of the source distribution.

postpone is a time period in minutes. After an alarm becomes active Big Sister waits for `postpone` minutes before it really sends out a message. If during this period the alarm is cleared it is silently dropped without a message. This is nearly the same as `delay`.

postpone_to is exactly the same as `postpone`. But the value is not exactly a time period in minutes - it is an absolute time of day, e.g.

```
postpone_to=06:00
```

will postpone alarms to 6 am. **Note** that the time is in 24h notation so 8pm for instance is 20:00, **not** 08:00pm.

Once you have understand alarming to this point there are a few challenges left for the advanced user: conditional alarming (`check` setting), using settings like priorities in pre-conditions and a few more. These are beyond the scope of this document and you will find them documented in the `CONFIG` file.

I did not understand anything at all

You are **not** alone. For all those who did not understand the consequences of the last few sections this section will contain a few more or less useful examples.

2.3 Advanced Configuration

2.3.1 Server security

The Big Sister server accepts client connection and therefore has some potential to be exploited by a hacker. Overall the server is considered rather secure since

- it is written in Perl and therefore does not suffer from buffer overrun problems
- it uses a simple protocol and therefore validity checking on incoming requests is rather trivial
- the available functionality is rather limitted

Anyway nobody will guarantee that there is absolutely no way of hacking Big Sister! A nasty user can at least annoy you by generating masses of false alarms, faking wrong system status, injecting wrong history data, etc.

Server Access

First of all you should limit access to the Big Sister server to those who really need access. You will need to configure your `adm/permissions` file in a reasonable way. The default `permissions` file

```
host .* => +all
```

makes Big Sister listen to everyone in the world speaking the right language! So you should start stripping this down a little.

As you probably already guessed each line in the `permissions` file contains a rule defining which clients are authorized to do special things. Every line contains the string `=>`. On the left side of this separator you will find a pattern telling Big Sister to what clients this rule applies, on the right side you will find a list of features this clients are allowed access to.

A client is identified in two ways:

- by the host it is running on
- by the user it is authenticating as

The second identification is not implemented yet - this makes less secure but at the same time things get really easy.

There are only a few ways how to identify a host: Either you identify it by its name or by its IP address. To make rules more powerful you can use wildcards in both cases. For instance

```
    host 192.168..* => +all
```

will permit access to every host in the 192.168 network, while

```
    host .*\.microsoft\.com => +all
```

will do the same for every host in the microsoft.com domain. **Note** that the strings are perl regular expressions which are slightly different from any other wildcarded patterns!

To clarify if you are trying to match a name or an IP address it is always a good idea to use `ip` or `name` instead of `host`:

```
    ip 192\.168\..* => +all
    name .*\.microsoft\.com => +all
```

There are a few more recognized keywords apart from `host`, `ip` and `name`. E.g. the server supports per user or per group access permissions - nevertheless there is currently no client supporting user authentication so it is sufficient for now to know the rules applying to hosts.

It is time now to understand what the expression at the right hand side of `=>` does mean: mainly it describes what server functionality the respective clients are allowed ("+") or disallowed ("-") to use. Apart from `all` (which means acess to all functions) Big Sister understands a few more function groups:

none access to nothing at all. Note that this works as a kind of "no operation", especially `+none` does **not** mean "access to none of the services"!

authenticate accept user authentication

status accept status information

grouping accept grouping information (see 2.3.2

page accept page request (compatibility with Big Brother only)

archiving accept history archiving requests.

alarm_acking accept alarm acknowledging. The alarm web interface makes use of this function

perf accept performance (trend) data

From most agents you will probably accept `status`, `grouping` and `perf` requests, while you should additionally accept `alarm_acking` from the Big Sister server. If your agents are located in the networks 192.168.1 and 131.156, and your Big Sister server's name is `bsdisplay` your configuration might therefore look something like:

```
host .* => -all
ip 192\.168\.1\..* => +status +grouping +perf
ip 131\.156\..* => +status +grouping +perf
name bsdisplay => +alarm_acking
ip 127\.0\.0\.1 => +status +grouping +perf +alarm_acking
```

The first line sets default permissions to "no access". The second through third line adds status, grouping and performance data access for agents in the listed networks. The last two lines add alarm acknowledging access for agents running on the server. In the 4th line we assume the server is also located in one of the networks listed in the second and third line and therefore already got access to status, grouping and performance data collection.

As you can see in the example above Big Sister will always go through the whole file and respects all the rules in the order they appear.

Therefore in the example above clients outside of 192.168.1 and 131.156 networks will get no access at all since only the very first rule (telling Big Sister to refuse access) applies.

Note: Host level authentication relies on the IP address of the systems connecting to the server only. Big Sister does not know what type of program is connecting. Therefore anyone having access to a respective system automatically can do to

Big Sister what an agent may do (e.g. by simply connecting to the server via the `telnet` command and typing in client/server commands). Big Sister is meant to work in environments where users usually are not expected to be interested enough in fighting against system administration to fake agent connections.

2.3.2 Dynamic grouping

2.3.3 Graphical status displays

2.3.4 Monitor modules

2.3.5 Performance data collection

2.3.6 Interlinking multiple Big Sister servers

2.3.7 SNMP support

2.3.8 SLA / Availability Reporting

Installation

The reporting module is included in the main distribution now. It is installed as part of

```
    make install
```

or with

```
    make install-reporting
```

Overall operation

There are two approaches to understanding reporting. First of all, there are two fundamental commands called `report_read` and `report_consolidate`. `report_read` reads in a specified file with status information or dependency rules for one specific day and stores the results in `var/reportdb/day-yyyy-mm-dd.statuslog`. Multiple runs of `report_read` – most probably you will at least read in the display.history and some dependency definitions - may incrementally update the same file in var/reportdb. One of the functions of `report_read` is the “Cumulator” which builds var/reportdb/*cumu* files out of the statuslog files by applying service hours / holidays definitions to the status information. Usually you will have multiple cumu files since you are interested in seeing statistics for multiple service levels. The files generated by `report_read` are actually of minor interest to you – they just serve as a kind of cache in order to reduce time spend in daily statistics operation. Another effect of this caching is that you do not need storing status information (display history) for the whole time period you are interested in.

`report_consolidate` then will go through all the cumu files in a specified time period, sum up time spent in specific status and create report files

```
var/reportdb/day-yyyy-mm-dd.statistics.cumuclass.name
```

for each cumulated file (textttreport_read) and each defined time period. These files are what you actually want to get.

Setting it up

Theory sounds rather complex, doesn’t it? Let’s go ahead to the real world then. In order to simplify the use of the reporting module an additional command `report_day` has been added. Usually you will just forget about `report_read` and `report_consolidate` and just use `report_day` which tries to do a sensible mixture of `report_read`s and `report_consolidate`s itself. As its name implies `report_day` is meant to be run on a daily basis. It will build statistics based on the following files:

```
var/display.history.*
  Big Sister’s status history

adm/reporting/servicehours
  defining when systems are
```

```
      expected to be working

    adm/reporting/holidays
      defining holidays (aka. time
      periods when systems are not
      expected to be in service)

    adm/reporting/cumulators
      defining which service levels
      should be reported (linking
      servicehours and holidays
      to status information)

    adm/reporting/override
      is meant to store manually
      maintained status information
      overriding display.history

    adm/reporting/dependencies
      telling us which services
      should be watched and how
      they depend on status
      information in display.history

    adm/reporting/statistics
      defining which time periods
      should be consolidated and
      what exactly we would like
      to see in the resulting
      report
```

This sound more exhaustive than it actually is – do not be afraid.

Best is to start with the simple things: servicehours and holidays. These files are rather self explaning – in servicehours you can define “in service” hours for each day of the week, while in holidays you can exclude whole days from service time. Note that the first column in each of these files contain a “class” specifier. This allows to define multiple levels of service – e.g. some systems might to be expected to run 24 hours 7 days a week while others are only expected to run from 8:00 till

17:00 on Monday through Friday. Define classes for all these service levels.

Servicehours and holidays will not be effective on their own. The rules actually linking service levels with status information are listed in the cumulators file. In this file you actually define your service levels based on servicehours and holidays. Each rule looks like

```
    levelname = service:class > holidays:class
```

Do not mind if you do not really understand what exactly “>” means. “levelname” is a symbolic name you can freely choose - at the very end you will get report files carrying “levelname” in their names.

It is time now for the more complex things: dependencies. In a real world you are usually not interested in seeing statistics for simple things like myserver.conn or myserver.smtp – in the SMTP case for instance you probably have multiple redundant mail servers increasing the overall mail service availability. So the mail service is available if any of your mail servers is up and running. In the dependencies file you tell the reporting modules which dependencies apply to your systems. The above rule would maybe look like:

```
    mailservice = history:myserver1.smtp | history:myserver2.smtp
```

telling that mailservice is up if at least one of myserver1.smtp and myserver2.smtp is up. Note the leading “history:”. Every information holder’s name in the reporting tool is preceeded by a prefix like e.g.:

history: – display.history information

service: – servicehours information

comp: – dependencies

holidays: – holidays information

override: – override information

To show a more complex example: let’s assume that the mail servers above depend on DNS to work correctly. So you define a DNS rule (let’s assume you run two redundant DNS servers dns1 and dns2):

```
    dns = history:dns1.dns | history:dns2.dns
```

Now you can use the result of the dns rule to make your mailservice rule more realistic:

```
    mailservice = comp:dns & \
        (history:myserver1.smtp | history:myserver2.smtp)
```

do not be fooled by the fact that the rule is now put on two lines – this is just for readability (the at the end of line one tells the reporting module the rule will be continued on the next line). Note that “dns” is in fact “comp:dns” – dependencies get automatically prefixed with “comp:”.

Note that the resulting statistics will only contain services defined via the dependencies mechanism. Also, dependencies with names starting with “_” are not appearing in the final output file – you can use such names as “internal variables”.

Of course some times your monitor will fail and report nonsense (e.g. because your agent or server got cut off). In this case you need a means for manually correcting such mistakes. This is done via the “override” mechanism. E.g. add the following line to the override file:

```
    mailservice,28.6.01 12:00,29.6.01 09:00,green,I know it worked
```

saying that mailservice was completely ok from June 28 12:00 till June 29 09:00. This line on its own does not yet change the statistics result. You have to set up your dependencies accordingly. The full mailservice rule should then look something like

```
    mailservice = comp:dns & \
        (history:myserver1.smtp | history:myserver2.smtp) \
        > override:mailservice
```

The “>” (override) operator tells us to prefer the expression on the right side of “>” to the expression on the left side for the whole time period(s) the expression on the right side is defined. In other words: whenever override:mailservice is defined the whole expression basing on history status is ignored and overriden by override:mailservice – actually this is probably what you guessed a long time before.

There is only one thing left you need to setup before we can get the first statistics: the statistics file. In this file you specify what “things” should be reported and which time periods the statistics should cover. As for now we can go along with the default file. It does report

```
Down           = time (secs) a service was red
N/A            = time (secs) a service was purple
Up             = time (secs) a service was yellow
 or green
Planned Outage = time (secs) a service was white
Service Hours  = time (secs) a service should have
                 been in service (according to the
                 cumulators file)
Down %         = Down/Service Hours
Up %           = Up/Service Hours
Availability   = The service’s availability
```

Note again that only dependencies (names starting with comp:) will appear in the final output. So as long as you do not define dependencies you will get no results. A dependency can of course be as simple as

```
comp:myrouter = history:myrouter.conn
```

Have you got the idea?

Running the reporting tool

The easiest way to run the reporting tool is via the `report_day` wrapper, e.g.:

```
bin/report_day
```

`report_day` is meant to be run every day and will by default read in status history / servicehours / holidays for the last two days and dependencies / cumulators / overrides for the last 10 days.

When running bin/textttreport_day for the first time you might want to compute statistics for a longer time back, do it with e.g.:

```
bin/report_day -h 30 -c 30
```

(-h: read history / servicehours / holidays 30 days back, -c: read the rest of the files also 30 days back)

You will see that this will take some time because `report_day` will compute each individual day separately. This is very inefficient for long time periods since the whole reporting system is optimized for daily (incremental) use.

Where are the results

You will get a bunch of files in var/reportdb. The most interesting files are day*.statistics.class.period. E.g. a file

```
day-2001-06-28.statistics.level1.oneweek
```

contains the statistic for the time from June 22 to June 28 2001 and the service level level1.

Note that though you defined oneweek to consolidate status for 7 days the reporting tool will create a statistics file for every day containing the last 7 days. This is intentionally setup like that. Probably you are only interested in one of these files per week though.

And what if I use savelogs/archivelogs

The reporting tool is compatible with savelogs/archivelogs (see bsadmin). Actually the display.history reader will not only read in display.history but also display.history.* files. If you archive your logs outside of var/display.history.* this will even drastically improve the efficiency of the daily incremental report generation steps since it is actually sufficient to have display history files available for 3 days back only!

Some of the file formats look very funny

Some of the file formats are actually CSV (well, more or less – do *not* try to use commas in cells!). That’s why they look rather unfriendly to a vi user. The file formats where defined with the idea in mind that someone might use some spreadsheet or database application for editing or importing/exporting the files. CSV is one of the formats most applications understand.

I need more than this – what shall I do

The reporting tool should be modular enough to add support for more log files or config files. For instance it would be nice to have some sort of database for the “overrides” instead of a flat table or at least some web based frontend. Also one could think about importing raw data from other monitoring systems.

One first thing we attempt to improve is the presentation of the results. E.g. we could set up some nice RRD graphs allowing us to compare current time periods with past time periods. Some means of inserting current statistics results on image maps would also be nice.

Please feel free to send suggestions or patches to the developers via

```
http://bigsister.graeff.com/
```