CS–TN–38

Local System Configuration for Syssies

Paul Anderson

All configuration information for departmental workstations is held in a single database, which is currently implemented as the NIS map `lcfg`. Each workstation has a list of *resources* in the database and this is the only information which is unique to that workstation; the workstation can be reconstructed automatically, and completely, from these resources and the generic operating system and local binaries. This allows machines to be rebuilt quickly after a hardware crash or an operating system upgrade, and it also permits validation and automatic generation of machine configurations.

Each software subsystem on the machine (eg. the print service) is controlled by an *object*. Selected objects can be started automatically when the machine boots, or run at regular intervals from `cron`. This allows different subsystems to be installed and maintained independently. The objects also respond to a standard set of *methods* (arguments) and print messages in a standard format; this allows them to be manipulated remotely with a special *object manager*.

Each object has its own set of resources in the resource database and its own log and configuration files. Every object is a member of a unique *class* (defined by the `class` resource), and the object methods are implemented by the *class script*. These scripts are written in a standard way and make use of a common set of functions which provide facilities such as reading resource values from the database.

1 Resources

A resource is a key-value pair with the form :-

⟨*host*⟩.⟨*object*⟩.⟨*resource*⟩ ⟨*value*⟩

A host name of `*` is a default value which is used if there is no explicit entry for a particular host. Resource names appear as shell variables within the class script and must hence conform to shell variable name syntax. Care is also required with resource values that contain shell meta-characters such as quotes.

Some resources are described as *read-only* in the class documentation. The values of these resources are set by the objects themselves; they do not appear in the database and cannot be changed (for example, the process id of some daemon process).

Resource values for a particular host can be changed using :-

➜ yped lcfg/⟨*hostname*⟩

Default resource values for an object are stored in files with an extension of `.def` and can be edited with :-

➜ yped lcfg/⟨*object*⟩.def.

Both host and object resources file are passed through the C preprocessor before incorporation into the database. Any common header files must have an extension of `.h`. The resource ⟨*host*⟩`.*.includes` is automatically generated for each host listing the header files which are included by that host.

2 Class Scripts

All the class scripts are contained in `/usr/local/lcfg/obj`. The class scripts can be invoked directly, but they should normally be run by using the object manager:

➜ om ⟨*host*⟩.⟨*object*⟩ ⟨*method*⟩ ⟨*args*⟩ ...

This allows execution from anywhere on the network by any authorised user (there is no need to be `root`). The object manager also reformats some of the output into a more readable form.

Each object responds to a standard set of methods :-

start - this is normally called when the object is started at boot time. It reads all the required resources from the database and starts any daemon processes. Configuration files which might normally be hardwired into the root file-system are typically created on-the-fly at this time, by using generic configuration files from the global file-system and machine-specific configuration information, from the database. If the object has already been started, then it is first stopped, allowing daemons to be restarted manually using this method.

stop - this is called to stop the object, either manually, or automatically when the system moves to a lower run level. It kills any running daemons, destroys the local copies of the resources that were read from the database, and deletes any temporary files.

run - this method is executed if the object is run from `cron` or if a particular process requires running manually; for example, updating a file-system, or applying new patches. The object must have been previously started. Note that the resources are not normally read from the database at this time; the values used will be those in effect at the time the object was started.

log - this method prints out any log from the object in a standard format. All log information should be made available in this way so that it can be read remotely, when the local log file may not be accessible. An extra optional numeric argument allows access to old (cycled) versions of the log.

cleanlog - his method is called regularly by `cron` to cycle the log files.

query - this method prints out the values of the resources at the time the object was started, together with other status information. The information is printed in a standard way so that it can be parsed and interpreted by the remote object manager.

doc - his method prints out the documentation for the object in a standard form. This is used to provide online help in the object manager and to generate the object summaries contained in the appendix.

The `methods` resource for each object lists the supported methods, which may include additional object-specific methods. For each method, the `auth_`⟨*method*⟩ resource specifies the users who are allowed to execute the specified method from the object manager. This is a space-separated list of items. If the client user/machine matches any of these items, then execution of the method is permitted. Each item can be a username, the wildcard `*`, a netgroup of machines, or a netgroup of users. These may be combined with the operator &.

3 Booting

The root file-system contains a very small number of local modifications which are sufficient to start `amd` and make the global file-system available. When the system boots, it starts the `boot` object which starts `amd` and all the other objects which need starting at boot time. The list of additional objects to be started is specified in the configuration resource `boot.services`.

On the first reboot after an installation, `boot` starts the `install` object which completes any local installation such as loading local binaries and adding clients to a server.

Some of the objects that are started at boot time may request a reboot of the system (for example, the `patch` object may have

applied new patches). In this case, `boot` reboots the system immediately. Objects request a reboot by writing a comment into one of the files specified in the `boot.bootlist` resource.

If a system is intended to carry its own local binaries, then it may need to boot initially using a remote `amd` cluster until it has loaded the local binaries onto its own disks. The resource `amd.install_cluster` specifies a cluster to be used for the first time that machine boots after an installation.

4 Updating Files on the Local File Systems

Three objects update files on the local disks :-

update - this ensures that the small number of local files installed on the root file-system are kept up to date by copying them from `/usr/local/lcfg/templates/`⟨*os-version*⟩.

patch - this automatically applies new patches to system.

updatelf - this runs on binary servers to update the local binaries using `lfu`.

Normally, `update` and `patch` are started by `boot` to ensure that the root is up to date whenever the machine boots. If any changes are made to the root file-system, then the machine is immediately rebooted. `updatelf` should also be started by `boot` on any machine which contains local binaries – this simply reads the resources and prepares a program script for `lfu` – it does not perform the actual update at boot time.

To keep file-systems up to date between reboots, `boot` should also be run periodically from `cron`[1]

[1]This can also be run on demand simply by executing :-
➜ om ⟨*hostname*⟩.boot run

In this case, `boot` is executed with the `run` method and it runs the objects specified in the `boot.run` resource. Usually, this will include `update`, `patch` and (on a binary server) `updatelf`. If any of these objects requests a reboot, then `boot` will either notify the person responsible for the machine, or reboot automatically, depending on the `boot.notify` and `boot.reboot` resources. `updatelf` may also change some files which do not require a complete reboot, but do require users to log out (eg. a change to `bash`). In this case, all users currently logged on to the system are notified.

In general, the cron jobs which update servers should run before the jobs which update their clients. This prevents clients attempting to install patches which have not been applied to their server.

5 Diskfull Installation

The local installation procedure interfaces with Sun auto-install to provide a completely automatic installation which installs Solaris, followed by all the necessary local modifications. Server, standalone and data-less machines can all be installed using the following process, with different resources for auto-install parameters :-

- Enter the machine in the NIS `ethers` map.
- Enter the machihine in the NIS `hosts` map.
- Create a set of resources for the machine in the NIS `lcfg` map, by referring to the defaults and the object documentation. (Copying a similar machine to use as a starting point is often useful).
- Boot the machine :-

  ➜ boot cdrom - install

  Or, on machines with old PROMS :-

➜ boot sd(0,6,2) - install

The machine should then install as follows :-

- The `install.local` defines a `bootparams` entry telling auto-install where to obtain the install configuration directory (Usually `/usr/local/lcfg/auto_install`).

- When auto-install runs, it executes the `begin` script in the configuration directory and this script generates the auto-install configuration information on-the-fly by reading resources from the `lcfg` map and creating corresponding entries for auto-install. Most of these resources correspond exactly to the documented configuration file entries for auto-install.

- After the Solaris installation is completed, auto-install runs the `finish` script. This mounts `/usr/local` from the server specified in the `install.local` resource and calls the `update` object with a special install method to copy the local changes onto the root of the newly installed system.

- When the new system first boots, the `install` object is started to complete any local installation that requires the local file-systems to be present, such as loading local binaries into `/disk/local`. The system then reboots if necessary to mount its own binaries.

  If the manual Sun-install screen appears, then auto-install has failed to mount the directory specified as `install.local` for some reason. If the Solaris installation completes, but the local modifications are not installed, then the logs in `/var/sadm/begin.log` and `/var/sadm/finish.log` should indicate the source of the problem.

6 Diskless Installation

As of Solaris 2.4, diskless workstations are not supported.

7 Network Install Servers

The disk-full installation described above requires the Solaris master disk to be available in the local CDROM drive. However, a server can be designated as an *install server*, in which case machines on the same wire can be installed without a CDROM. When the server is installed (from CDROM) it can be set up as an install server by specifying the `install.install_server` resource. This requires an `/export/install` partition of sufficient size.

To enable a client to install from this install server :-

- Enter the machine in the NIS `ethers` map.
- Enter the machine in the NIS `hosts` map.
- Create the resources for the machine in the NIS `lcfg` map, including :-

  root - normally ⟨*server*⟩`:/export/root/`⟨*client*⟩.

  install - normally ⟨*server*⟩`:/export/install`.

  arch - the kernel architecture.

- Ensure that the `system_type` is *not* specified as `client`.
- Install the client on the server :-

  ➜ om ⟨*server*⟩.install add ⟨*client*⟩

- Boot the client :-

  ➜ boot net - install

The installation should then proceed in the same way as an installation from the CDROM.

8 X Terminal Installation

X Terminals can also be installed using the install add method. Currently, support is provided for Sun3 and Sun4 workstations configured as X terminals, as well as NCD X terminals. These require some `install` resources, and may also be provided with `info` resources.

To install an X terminal :-

- Enter the machine in the NIS `ethers` map.
- Enter the machine in the NIS `hosts` map.
- Create the resources for the terminal in the NIS `lcfg` map.
- Make sure that the server is running `bootps` and `tftpd` in the `inet` services.
- Make sure that the `queryhost` resource is defined and the specified host is running the `xdm` object in the `boot` services.
- Make sure that the `fshost` is defined and the specified host is running the `Fontserver` object in the `boot` services.
- Install the terminal on the server :-

  ➜ om ⟨*server*⟩.install add ⟨*client*⟩

- Boot the X terminal.

The `swap` and `screen_type` resources are only needed for Sun terminals and the `nameserver` resource is only applicable to NCD terminals. If no `swap` resource is specified, then a non-swapping kernel will be used. Not all combinations of architectures and/or swapping/no swapping are supported. The `arch` resource determines the sun kernel architecture (if it starts with `sun`) or the NCD server code file to be used (if it starts with `ncd`)

The directory `/usr/local/share/sun-xterm` contains the boot files and the template root directories for Sun X terminals.

XDM can be run on a workstation console by setting `xdm.servers` to an appropriate value (see the `xdm` man page) and disabling console logins by removing `console` from the `saf.logins` resource.

9 Writing New Class Scripts

The source for all the class scripts is in `/home/lcfg/export/obj` where it is maintained under `rcs`. All scripts source the `generic` script which provides a set of utility functions and generic methods that can be inherited directly by many classes. This allows simple classes to be be created with very little code. The `SAMPLE` class is provided as a template for writing new classes

The standard output from the objects must conform to a standard format so that it can be parsed by the object manager and other utilities. The functions provided in the `generic` script are the most convenient way of ensuring this.

Some conventional directories are used for various files :-

/usr/local/lcfg/conf - contains generic configuration files and read-only data files.

/var/obj/conf - contains generated configuration files and other writable files that must persist between object invocations.

/var/obj/log - contains log files. Normally, the name of the log file is predefined as `$logfile` and this directory will not be accessed explicitly.

/var/obj/status - contains status files. Normally, the name of the logfile is predefined as `$statusfile` and this directory will not be accessed explicitly.

/var/obj/tmp - contains any temporary files created by the object. These are destroyed when the object stops or when the machine reboots.

Filenames in these directories should generally have the form :-

⟨*object-name*⟩.⟨*something*⟩

The `doc` method should print the documentation in troff source format. This should start with a description of the object, followed by a description of any resources. The macro `.Re` should be used to introduce a resource description; the first parameter is the name of the resource and the second parameter is `R` or `W` depending on whether the resource is read-only or writable. The macro `.Ce` sets the first argument in courier font which should be used for all references to resources, filenames and other items which may be typed literally at the keyboard.

Default objects of the new class can be defined by creating a file with the same name as the required object and an extension of `.def` in the directory containing the resource files on the YP master server. This file should contain the default resources for the object (including the `class` resource). One default object with the same name as the class should always be defined.

10 Changing the Root Template

As few changes as possible are made to the standard root file-system. This minimises the frequency with which the files on the root must be updated. `/usr/local/lcfg/templates/`⟨*os-version*⟩ contains hierarchies of files which must be copied onto the root file-system during a new installation. The `update` object also runs regularly to ensure that these files are up-to-date. Files can be installed onto the root simply by copying them into the template. `update` will rename any existing copy of the file to `+ORIG:`⟨*file-name*⟩ and copy in the file from the template. Files can be deleted from the root by creating a file with the name

`+DELETE:`⟨*file-name*⟩ in the template. Files in the template with a name of the form `+HUSH:`⟨*file-name*⟩ inhibit the usual request for a machine reboot that would be generated whenever the file ⟨*file-name*⟩ is changed.

`Update` also copies a selected set of objects from the global file-system into `/etc/obj`. The `update.objlist` resource specifies the objects to be copied.

11 The Object Manager

The daemon `omd` runs on every machine to service requests from the object manager client `om`. `omd` uses RFC931 to authenticate requests which are only permitted if the user is specified in the appropriate resources. Valid requests are executed and logged.

The current client program is just a prototype which submits simple requests of the form :-

➜ om ⟨*host*⟩.⟨*object*⟩ ⟨*method*⟩ ⟨*args*⟩ ...

Output from the object is displayed directly with very little editing and error handling is basic.

Eventually, it is hoped that `om` will be able to submit requests to groups of machines in parallel and perform more intelligent processing of the output from the objects. Better authentication (perhaps Kerberos) should also be used.

Revision History

This document is published as technical note number 38 by the Department of Computer Science at the University of Edinburgh. The document's revision history is as follows:

- Revision 1.2 (14th October 1996).
- Revision 1.12 (5th November 1995).
- Initial release (November 1995).

Appendix: Classes

This appendix lists the currently implemented object classes. Current documentation for a class can be obtianed by calling the doc method of the class script.

Object: FontServer

Version: 1.6
Author: gdmr
Last Modified by: paul

Description:

This object starts the `FontServer` daemon. (`xdm` is started from its own object.)

Object: accounting

Version: 1.11
Author: gdmr

Description:

This object does process accounting. The "start" method turns on accounting. The "stop" method turns it off. The object should be "run" from cron once every hour, when it will either do the nightly and monthly processing or checkpoint the recording files. In the second half of the month the object creates the "do_month" indicator file every night; in the first half it checks for the existence of the file and removes it and runs the monthly accounting if it exists. Thus the machine doesn't have to be up on any particular day for the monthly accounting to be run.

Resources:

hour : The hour at which the processing should be done. The default is 01 (i.e. 1am). Note that the leading zero is required. Every other hour the recording files are checkpointed.

save_directory : If non-null, specifies a directory to which the recording files are copied before being processed and destroyed by the regular nightly processing. Both system activity files as generated by sar, if any, and process accounting files are copied. Note that sar may be run independently of process accounting, but if it is then some way will be needed to initialise each day's recording.

varobjlog_base_list :

varobjlog_extra_list : Logfiles from /var/obj/log which should be archived along with the accounting information. "base" should be set globally, with "extra" being per-host additions.

Object: amanda

Version: 1.9
Author: iro

Description:

This object uses the `AMANDA` package to handle backing up disks, and sets up the necessary configuration files, `inetd` `services` and `crontab` entries. See the inet object and the services NIS map for the first two, the cron object for the crontab stuff. Running this object with an argument of 1 causes a check on the tape currently in the drive (or not) to be made, and mail is sent to the amanda operator; this works on weekdays only.

Resources:

cluster : This resource determines which configuration files are run on a master host. No dumps are done on Sundays. See "man amanda" for further details.

Object: amd

Version: 1.30
Author: paul
Also modified by: dwb,gdmr

Description:

This object starts the `amd` automounter.

Resources:

cluster : The `amd` cluster variable.

install_cluster : This cluster is used instead of `cluster` the first time that the system boots after an installation. This is necessary for systems which normally hold their own local binaries, since these will not be loaded when the system first boots after an install. If this option is set, `amd` also requests a reboot.

maps : A (space-separated) list of map names and corresponding mount points. If this resource is missing, a default list of maps is taken from the `master.hesiod` NIS map.

key : The name of the key in the `master.hesiod` NIS map which is used to look up the default map configuration.

options : A list of `amd` options.

waitfs : After starting `amd`, the object waits for this filesystem to become available before continuing. This ensures that all the necessary files are available for any objects which start after `amd`.

usr_local_links :

disk_sun451 :

disk_share : These three resources control whether a real /usr/local directory full of symlinks is created at start/run time. If usr_local_links is set then /usr/local is not automounted. Instead, disk_sun451 and disk_share are used, together with the cluster, to build a directory full of real symlinks pointing directly to the disc if possible, else to /usr/remote.

homemap : If this resource is present, it is assumed to be a subdirectory of `/disk/home`. An amd map is created in the file `/var/obj/tmp/amd.homemap` which is suitable for referencing in the `maps` resource to create a configuration with local home directories.

Object: annex

Version: 1.4
Author: ajs

Description:

This object configures the YP. It doesn't directly control any daemons as these have to be started before any of the "lcfg" objects. It transfers YP maps to make a machine a slave server and manipulates the YP bindings so a machine can be bound to a particular server.

Resources:

type : The type of YP service on this machine. Valid options are `client` and `slave`.

servers : A list of servers to `ypbind` to. Specifying no servers forces `ypbind` to broadcast for it's YP maps. `master`W The YP master from which the YP maps are sucked.

Object: apache

Version: 1.11
Author: paul

Description:

This object runs the Apache WWW server.

Resources:

configfile : The pathname of the configuration file.

Object: auth

Version: 1.29
Author: paul
Also modified by: gdmr

Description:

This object constructs all the authorization files allowing access to the machine. This includes `/etc/passwd`, `/etc/shadow`, `/etc/group`, `/etc/hosts.equiv` and `/.rhosts`.

Resources:

rootpwd : The encrypted root password. This resource is also used by the `install` object to set the initial root password after an installation.

users : A (space-separated) list of users or netgroups to be added to the base password file. The password file information is taken from the NIS map. The base password file is `/usr/local/lcfg/conf/passwd` and the base shadow file is `/usr/local/lcfg/conf/shadow`.

groups : A (space-separated) list of groups to be added to the base group file. The base group file is `/usr/local/lcfg/conf/group`.

rhosts : A (space-separated) list of items to be added to the `/.rhosts` file.

equiv : A (space-separated) list of items to be added to the `hosts.equiv` file.

message : This resource specifies the message to be displayed to unauthorized users attempting to log on to the system. If

the resource starts with a '/', it is assumed to be the name of a file containing the message.

noauth_shell : This specifies the shell to be used for users that are not allowed to log in. The special shell `/usr/local/lcfg/bin/ftponly` is the same as the default shell `/usr/local/lcfg/bin/noauth` except that it appears in `/etc/shells` and thus allows users access via ftp.

Object: boot

Version: 1.41
Author: paul
Last Modified by: dwb
Also modified by: gdmr

Description:

This object starts all the objects which should be started automatically when the system is booted. `amd` is automatically started first and should not be specified expilicity in the `services` resource. Objects specified in the `services` resource are then started in order. Some objects (currently just `dns`) must be started before `amd` and should not be specified. If this is the first boot after an installation, then the `install` object is started automatically after all the other objects. If any of the objects request a reboot (by touching files named in the `boot.bootlist` resource) then the system is rebooted. The local filesystem can be assumed to be available by the time that the specified services are started.

The `run` method runs the objects specified by the `run` resource. If any of the objects request a reboot (by touching files named in the `boot.bootlist` resource) then the system is rebooted (if the `run_reboot` resource is non-null), and the user specified by the `notify` resource is mailed. If any of the objects have requested a logout (by touching files named in the `boot.keeplist` resource) then all logged in users will be notified. This is designed to be run from `cron` to periodically update the filesystem by running objects such as `update`, `patch` and `updatelf`. Before running the specified objects, `boot` calls the `cleanlog` method to cycle the logfiles for any objects listed in the `cleanlog` resource. The boot object also supports a number of resources which are used to generate the `bootparams` map.

Resources:

services : A (space-separated) list of the objects to be started (in order).

user_*service* : This is the username under which the object for the specified service will be run. By default, objects are run as root (assuming that root is running the `boot` object).

level_*service* : The run level at which to run the specified service. This must be `2` or `3`.

run : A list of objects to be run when the `boot` object is executed with the `run` method .

cleanlog : A list of objects to be called with the `cleanlog` method when `boot` is run. A '`*`' in the list of objects is substituted with a list of all objects specified in the `boot.services` resource.

run_reboot : If this resource is non-null, then the system is automatically rebooted if any object has requested a reboot.

notify : If this resource is non-null, then the specified user is mailed if any object has requested a reboot.

bootlist : Each time boot is run, it checks the files named in this resource. If any of them have changed, then boot uses the message contained in the changed file(s) as a reason for requesting a reboot of the machine.

keeplist : Each time boot is run, it checks the files named in this resource. If any of them have changed, then boot uses the message contained in the changed file(s) as a reason for requesting all current users to log out.

explain : If this resource is non-null, then information messages will be printed to the the console explaining the reason for any reboot when `boot` is started.

Object: cadence

Version: 1.9
Author: cc

Description:

This is the Cadence object. It starts a Cadence locking daemon called cdsd which needs to run on all home directory servers. (This is NOT the Cadence licence manager daemon.) If cadence.hspice is set to "yes" then it also runs the "metaserver" licence daemon for hspice.

Object: cap

Version: 1.18
Author: paul
Last Modified by: jst

Description:

This object constructs the uar.conf file and starts the `uar` (Unix Appletalk Router) daemon. If the `services` entry is non-null, it also starts `atis` and all the listed CAP services.

Resources:

services : A space-separated list of CAP services provided by this host. The services `uar` and `atis` should not be included in this list.

capzone : The Appletalk zone in which this machine's CAP services should appear, if different from the default zone for this wire.

capwire : The Appletalk network number for the interface on which this machine's CAP services should appear. If absent, no CAP services can be run, but the machine will still act as an Appletalk router.

excluded_wires : A comma-separated list of wires to exclude from routing even though they appear in the zone table.

Object: cap1

Version: 1.10
Author: jst
Last Modified by: paul

Description:

Each instance of this class is an individual user-visible CAP service, such as `lwsrv`. The class is usually started by the cap object.

The `run` method is only supported for the `macdump` instance.

Resources:

choosername : The name under which the object will be listed in the Macintosh Chooser.

Defaults to the hostname followed by the string `(Unix)`. Ignored for `lwsrv`, which takes its choosername from the name of the print queue (there may be more than one).

options : Extra options to be passed to the daemon.

afpvols : (AUFS only): the file which specifies the global mapping from Unix directories to Mac volumes.

Defaults to `/usr/local/share/cap/afpvols`.

typelist : (AUFS only): the file which specifies the global mapping from Unix filename extensions to Macintosh Finder types, and any translation that should be performed.

Defaults to `/usr/local/share/cap/afpfile`.

printers : (lwsrv only): a space-separated list of printers served off this host.

fontfile : (lwsrv only): the name of a file containing a list of fonts which the printer can be assumed to support.

Defaults to `/usr/local/share/cap/LWPlusFonts`.

messagefiles : (motd only): a space-separated list of files containing messages to be presented to the user.

configfile : (macdump only): the name of the configuration file indicating the order in which dumps are to be run. This file should be in macdumptab (5) format.

Object: copier

Version: 1.1
Author: iro

Description:

This object sets up a solaris system to handle management of copier accounts. All it does at the moment is to link /dev/copier to /dev/ttya on the selected host.

Object: cron

Version: 1.28
Author: paul
Last Modified by: gdmr
Also modified by: jtb

Description:

This object starts the `cron` daemon. Authorization files are constructed for `cron` and `at`. The `cron` object then deletes the existing crontab files for any users who have base crontab files in the directory specified by the `crontabs` resource, or who have an `additions` resource. Base crontabs are then copied in from the crontabs directory, and any additional entries specified by `additions` resources are added.

Resources:

allow : A (space-separated) list of users or netgroups for the `cron.allow` file.

deny : A (space-separated) list of users or netgroups for the `cron.deny` file.

atallow : A (space-separated) list of users or netgroups for the `at.allow` file.

atdeny : A (space-separated) list of users or netgroups for the `at.deny` file.

crontabs : A directory containing base crontabs. Any crontabs in this directory will replace the corresponding crontabs on the machine.

additions : A (space-separated) list of *tags* for additional crontab entries specified in the resource database.

add_*tag* : The crontab entry for the specified tag.

owner_*tag* : The username under which the crontab entry for the specified tag should be run.

objects : A space-separated list of objects to be run from `cron`. A `cron.run_`*`obj`* resource must be present for each object listed. The object is executed with the `run` method at the time specified by the `cron.run_`*`obj`* resource.

run_*obj* : The time at which to run the specified *object* (in `crontab` format).

user_*obj* : The username under which to run the specified *object*.

args_*obj* : Additional arguments to supply when running the specified *object*.

cronlog : If this resource is non-null, then cron logging is enabled, otherwise it is disabled.

path : This resource specifies the default `PATH` used for non-root cron jobs.

supath : This resource specifies the default `PATH` used for root cron jobs.

Object: cs2

Version: 1.18
Author: dwb

Description:

This is the CS2 monitoring object. The `lastlogins` method checks the last login time for all CS2 users according to their .last_login files. The `logcheck` method checks the last login times of a tutors tutees. The `run` method does the process accounting summary.

Object: dns

Version: 1.31
Author: paul
Last Modified by: gdmr

Description:

This object starts the DNS .

Resources:

type : The type of DNS service. Valid options are `client` (the default) and `server`.

servers : A list of servers to place in the `resolv.conf` file.

include : Defaults to null, and should not normally be set to anything. The name of a file to be INCLUDEd in the named.boot file.

forwarders : A list of forwarders.

options : A list of `resolv.conf` options.

proxy : A list of IP addresses which will be substituted for the text "PROXY" in the boot file template.

search : A list of domains for the `resolv.conf` "search" list.

global_sortlist :

cluster_sortlist :

local_sortlist : Sortlists to be included in the `resolv.conf` file. "local" entries come first, followed by the machine's attached wires, with the "global" entries coming last.

local_netmask : A netmask to be applied to the machine's attached interfaces when constructing the sortlist.

pending_pending_zonelist :

pending_final_zonelist : These resources control the "pending" update mechanism. There is normally no reason to change these from their default values.

bogus_reverse_zones :

bogus_reverse_zonefile : These resources are not normally used. They are there to provide a reasonably clean way to stop remote servers beating on us for a zone that we don't have. bogus_reverse_zones is a list of zones that we think are bogus; bogus_reverse_zonefile is the name of the zonefile that we use for these zones.

Object: elmd

Version: 1.5
Author: gdmr

Description:

This object starts the AutoCAD elmd licence daemon.

Object: flexlm

Version: 1.4
Author: jst
Last Modified by: paul

Description:

This starts Highland Software's FLEXlm (Flexible Licence Manager). The `start` method looks in `/usr/local/share/flexlm/servers` for one or more licence files naming the current host as a server, and starts one instance of `lmgrd` for every one found. It is assumed that the files in this directory have previously been arranged according to the port number on which `lmgrd` is expected to listen for licence requests.

Resources:

ports : A space-separated list of ports on which a listener has been started.

Object: gated

Version: 1.6
Author: gdmr

Description:

This object starts the gated . In addition to the usual resources the following are accepted:

Resources:

rip_accept_default : If set then the routes to the "default" network are accepted. If not set (the default) they are not.

rip_import : A list of networks, routes to which will be accepted. The default is for routes to EdLAN to be accepted. If a netmask other than the "natural" one is required it should be appended to the network number separated by a ':'.

static_hosts : A list of hosts to which static routes should be installed. If this is set then the "static_gateway" resource must also be set.

static_networks : A list of networks to which static routes should be installed. Again the "static_gateway" is required.

static_gateway : The address of the router through which static routes should be directed.

disable : If this resource is set then the system will revert to using routed/rdisc instead of gated.

config_file : The name of the configuration file which should be generated and fed to gated.

gated_binary : The name of the gated image itself.

Object: generic

Version: 1.72
Author: paul
Last Modified by: dwb
Also modified by: jtb,gdmr,jst

Description:

This object includes a default set of methods and resources for use by all other objects. It is not normally executed directly. The following resources are common to all objects. ..ds On *

Resources:

class : The object class. This defines the script which implements the object methods and the name to be used when locating defaults for resources which are not provided explicitly for the object itself.

debug : Non-null to enable debugging. The actual value may be passed as debugging options to any processes started by this object.

methods : A (space-separated) list of methods accepted by the object.

auth_*method* : A (space-separated) list of groups allowed to execute the specified method from the object manager. Each group may contain usernames, netgroups (of machines or users - preceeded by a '`@`') or the wildcard '`*`', combined with the operator '`&`'.

pid : A (space-separated) list of process ids for any processes started by this object.

time : The time that this object was last started.

status : The status of the object. This is normally `active` if the object has been started, and `inactive` when it has been stopped.

rcs_revision : The RCS revision number of the object.

rcs_author : The username of the last person to modify the object.

osname : The name of the operating system as given by "uname -s".

osrelease : The operating system release as given by "uname -r".

osversion : The version of the operating system as given by "uname -v".

cleanlog_logs : The list of log files to be cycled. If it isn't set the standard log file is cycled.

cleanlog_freq : The frequency with which the logs are cycled in days.

cleanlog_zap : The number of old logs to be kept.

logowner : The owner of the log files.

logfilter_filter : If set, the logfile is fed through the filter specified by this resource before it is cycled. Any output is mailed as specified.

logfilter_mailto :

logfilter_ccto : These resource control to whom the log filter output is mailed. logfilter_mailto ususally defaults to root in the object's ..def file.

Object: hlfs

Version: 1.7
Author: gdmr
Last Modified by: paul

Description:

This object starts the `hlfs` daemon.

Object: inet

Version: 1.27
Author: paul
Also modified by: jtb,gdmr

Description:

This object starts the `inetd` internet daemon. If the local filesystem is not available, then `inetd` is started with a minimal set of services from `/etc/inetd.conf.base`. If the local filesystem is available, then the services specified in the `defconfig` resource are also started. Finally, the `services` resource is used to specify the additional services that are to be started. The specification of these services is taken from `optconfig`. At boot time, `inetd` is started with the minimal set of services before `amd` runs, and then later restarted by `boot` to start the additional services.

Resources:

services : This resource is a (space-separated) list of additional services to be run. These services must appear in the file .

options_*service* : Additional command line arguments to be passed to the daemon for the specified service.

allow : A list of services that are to be included in the `/etc/hosts.allow` file for access control by `tcpd`. The configuration file must specify the `tcpd` wrapper program for these entries. If the service name starts with a lower-case letter, it is looked up the configuration file and the name of the corresponding daemon is entered in the `hosts.allow` file, otherwise, the service name itself is entered.

allow_*service* : The access control list for the specified service. This is a list of patterns as specified in '`man hosts_access`

deny : A list of services that are to be included in the `/etc/hosts.deny` file for access control by `tcpd`. The configuration file must specify the `tcpd` wrapper program for these entries. If the service name starts with a lower-case letter, it is looked up the configuration file and the name of the corresponding daemon is entered in the `hosts.deny` file, otherwise, the service name itself is entered.

deny_*service* : The access control list for the specified service. This is a list of patterns as specified in '`man hosts_access`

Object: info

Version: 1.12
Author: paul
Last Modified by: lcfg
Also modified by: dwb,iro

Description:

This object is currently used only to access the informational resources for a specified host. The method `get` *`host`* prints the `info` resources for the specified host (default is the current host). Some resources from the `install`) object are also used to generate the `system_type`, `server`, and `clients`. `doc` is the only other supported method. Eventually, this object may start a process to check the values of thos info resources which can be checked automatically.

Resources:

groups : A list of netgroups to which this machine should be added by `gengrp`. For each *group* listed in this resource, the primary machine name is added to the netgroup `HOSTS_`*`group`* and all the interfaces of the machine are added to the netgroup `IP_`*`group`*.

type : This identifies the host as private workstation (`private`), a public workstation (`public`), or a server (`server`).

owner : If the workstation is `private` then this resource is a (space-separated) list of usernames of the workstation owners. For other types of workstation, this is a description of the class of users (eg. CS1 students).

descr : A short description of the machine (in html). If this resource begins with a '/', then the resource is the name of a file containing the description.

make : The make of the workstation (eg. Sun).

model : The model of the workstation (eg. 10/50).

hostid : The host ID.

sno : The serial number.

location : The normal location (room number) of the machine.

memory : The memory size in Mb. This resource should be a (space-separated) list of values representing the size of individual SIMMs in the machine.

total_memory : The total memory memory size in Mb. This resource is the sum of the values listed in the `memory` resource.

instd : The date the host was installed

maintd : The name of the company maintaining the machine

kbd : The keyboard type

kbd_sno : The keyboard serial number

disks : A (space-separated) list of disk tags.

disksize_*tag* : A resource of this type specifies the size for each disk.

disktype_*tag* : A resource of this type specifies the type of each disk.

diskdev_*tag* : A resource of this type specifies the device for each disk.

diskinst_*tag* : A resource of this type specifies the installation date of each disk.

diskmaint_*tag* : A resource of this type specifies the maintainer for each disk.

displays : A (space-separated) list of display tags.

dpytype_*tag* : A resource of this type specifies the type (colour, mono or grayscale) of each display.

dpyres_*tag* : A resource of this type specifies the resoultion of each display.

dpydev_*tag* : A resource of this type specifies the device of each display.

dpysno_*tag* : A resource of this type specifies the serial numbers of each display.

Object: install

Version: 1.80
Author: paul
Also modified by: dwb,gdmr,jst

Description:

This object gets started automatically by `boot` the first time that the system is booted after an installation. `updatelf` is run if necessary to load binaries onto any local filesystems, and an install server is created, if required. Any clients specified in the `install.clients` resource are then installed.

Resources from this object are also used to build the profile used by auto-install.

This objects provides the additional methods `add` and `remove` which accept the name of a diskless client to be added to or removed from the local host as a server. If the specified client has a `system_type` resource which is not `server`, then the client will be installed as a network install client, rather than a diskless client. When the machine is installed, the ethernet interfaces are configured as specified in the `interfaces` resource. If the interfaces are later changed, then they can be reconfigured using the special `interfaces` method. The machine will require rebooting after reconfiguring the interfaces.

Resources:

system_type : This is the system type as documented for auto-install. The additional types mac,`client`,xterm, and `printer` are supported for disk-less clients, X terminals, Macintoshes, and printers respectively.

partitioning : This is the partitioning as documented for auto-install.

filesystems : A space-separated list of filesystem *tags*. An auto-install `filesys` entry is generated for each named filesystem. Each specified filesystem must have a corresponding `fs_`*`tag`* resource. Any filesystems with are specified as `free` must appear in this list *after* all other filesystems on the same disk.

fs_*tag* : The auto-install `filesys` entry for the specified filesystem.

usedisk : A list of disks to be used by auto-install.

dontuse : A list of disks to be ignored by auto-install.

num_clients : Number of clients used by auto-install.

client_arch : Client architectures used by auto-install.

client_swap : Client swap used by auto-install.

cluster : This is the software cluster to load, as documented for auto-install. For disk-less clients, this must be `SUNWCall` which makes available all the packages loaded onto the server (packages can be removed from this set with the `delete` resource).

swapsize : This is the swap space in Mb for disk-less clients.

add : This is a (space-separated) list of software packages to load in addition to the cluster. This must be null for disk-less clients.

delete : This is a (space-separated) list of software packages in the cluster to be ignored. The default list includes those packages normally installed in `/opt` which is usually mounted from the network. This resource is also used when installing clients to list packages which should not be installed in the client root.

local : This is the remote filesystem which is mounted as `/usr/local` to obtain the template and objects for installation (for example - *server*`:/disk/local/sun4-51`).

auth.rootpwd : This resource is taken from the `auth` object to initialize the root password at installation time.

install_server : If this resource is non-null, the host will be installed as an install server by copying the OS installation software from the CDROM into `/export/install`. This partition must exist and must be large enough for the image. The machine must also be installed directly from the CDROM rather than over the network. If this resource has the value `proxy`, Then the host will be installed as a proxy install server only.

server : The `install` entry for the `bootparams` map (for network installations). This should normally be *server*`:/export/install`.

root : The `root` entry for the `bootparams` map (for diskless clients and network installations). This should normally be *server*`:/export/root/`*client*.

swap : The `swap` entry for the `bootparams` map (for diskless clients). This normally specifies a swapfile on the server, (eg. *server*`:/export/swap/`*client*), but it may also specify the name of a local disk. (eg. `c0d0t3s2`).

dump : The `dump` entry for the `bootparams` map (for diskless clients).

arch : The kernel architecture.

updatelf : If this resource is non-null, the `updatelf` object is run when the system is installed to load the local filesystems. `updatelf` must be included in the `boot.services` resource.

clients : A list of client names to be installed on this server after the server itself is installed.

cautious : If set, any failures in installing clients will cause the whole installation to fail. If unset (the default) a warning will be given and the installation will proceed.

console : The type of the console terminal.

interfaces : A list of ethernet interface names. Each interface must have a corresponding `hostname_`*`interface`* resource.

hostname_*interface* : For each interface, this resource specifies the corresponding hostname.

bootserver : For a client, or an X terminal, this resource must specify the name of the bootserver. The client must also be added to the boot server by running `install add` on the `bootserver`.

queryhost : The host to which X terminals will speak XDMCP. If the address of this host changes after an X terminal has been installed, the X terminal must be re-installed. If this resource is missing, Sun X terminals will use the `bootserver` and NCD X terminals will display a menu.

fshost : The fontserver host for X terminals. If the address of this host changes after an X terminal has been installed, the X terminal must be re-installed. (default is the `bootserver`).

fsport : The port number used by x terminals to contact the fontserver.

nameserver : The DNS nameserver host for X terminals. If the address of this host changes after an X terminal has been installed, the X terminal must be re-installed. This is currently only required for NCD X terminals (default is the `bootserver`).

screen_type : This should be `colour` or `mono` or `grayscale`. This resource is currently only used by X terminals.

xserver_args : These arguments are passed to the X server in Sun X terminals. The default disables backing store.

ncd_configfile : This is the base configuration file used by NCD X terminals. It is passed through the C preprocessor to craete a host-specific configuration file.

Object: kerberos

Version: 2.3
Author: gdmr

Description:

This object starts the `kerberos` daemons.

Object: led

Version: 1.1
Author: paul

Description:

This is a dummy object which accepts the single method "edit" to call the led.pl script and edit symbols in the lcfg maps. This allows the script to be called by the object manager, allowing remote execution on the yp master with authorization control.

Object: locks

Version: 1.1
Author: root

Description:

This is the locks object. It starts the locks daemon which controls the card swipe door locks. It also cycles the log for the locks object.

Object: mail

Version: 1.25
Author: paul
Also modified by: dwb,gdmr

Description:

This objects constructs all the necessary configuration files and starts the mail daemon. The `queue` method can be used to look at the sendmail queue. The `kick` method can be used to process the sendmail queue.

Resources:

type : The type of mail service. Currently only `client` is supported.

mailprog : The name of a local sendmail program. This program is copied in place of `/usr/lib/sendmail` if it exists.

configfile : The `sendmail.cf` configuration file used by MUAs.

daemonconfig : The `sendmail.cf` configuration file used by the `sendmail` daemon which runs the queues.

inconfig : The `sendmail.cf` configuration file used by the `sendmail` which runs from the inet `smtp` service to handle inbound connections.

options : The `sendmail` options.

root : This resource is processed by the `root_redirect` script which runs on the mail host. It is also written to the file `/.forward`. Any mail directed to `root` on this host will be forwarded to the address specified by this resource.

popopts : This resource should contain a list of options for the POP daemon. These options are saved into `/var/obj/conf/pop.options` which is read by `kpopd` to locate the default options when it is started (by inetd). The most useful values are `k` for Kerberos authentication, and `u` for unix format mailboxes (default is MMDF).

varmail : If this resource is specified, any directory `/var/mail` is removed and `/var/mail` is linked to the directory named in this resource.

Object: mrouted

Version: 1.2
Author: ajs

Description:

This object configures and starts the mrouted daemon.

Resources:

tunnels : A list of tunnels to create. Each entry requires an associated value. Optional.

tunnel_X : The value to be assigned for X, where X is a tunnel listed in the

quired :

Object: multi

Version: 1.3
Author: dwb

Description:

This object starts up the licence daemon for the Waves package used by the Cog Sci summer school.

Resources:

waves_server : defines which host is the nominated waves server.

Object: news

Version: 1.26
Author: dwb

Description:

This is the news object, it tidies up the news system on reboot. The `expire` method can be used to start off a news expiry run. This takes a long time. The `add` method can be used to add a new newsgroup. The newsgroup name and flags should be given as paramaters. The `remove` method can be used to remove a newsgroup. The newsgroup name should be given as a parameter. The `modify` method can be used to modify the flags associated with a newsgroup. The newsgroup name and flags should be given as paramaters. The `newsflow` method generates the daily news flow report. Normally run daily from the cron. The `usage` method generates the daily report of news spool disk usage by news group. Normally run daily from the cron. The `locpostsum` method generates the daily report of local postings with distributions wider than DCS. Nornally run daily from the cron. The `oldcomp` method performs the compression of archived news articles in the old news spool area. Normally run weekly from the cron. The `watch` method performs the standard C-news status check. Normally run a few times a day from the cron. The `daily` method performs the daily log shuffling and generates some of the daily reports. The `newexplist` method generates a new explist file for use by the `expire` method. This is normally run weekly from the cron. The `run` method can also be used to start an expiry run by giving it ***`expire`*** as an argument.

Object: nfs

Version: 1.14
Author: paul
Last Modified by: dwb
Also modified by: gdmr

Description:

This object starts the NFS service. `nfs` is not started by the `boot` object because it is not active in run level 2. It is started on entry to run level 3. Exported directories are created if they do not exist. The `run` method scans all exported filesystems for `.nfs` files, deleting those which are older than one week.

Resources:

dfstab : The name of the file containing the system filesystems to export.

exports : A list of filesystem tags to export, in addition to the filesystems exported from the `dfstab`.

fs_*tag* : The pathname of the filesystem to export for this tag.

options_*tag* : The mount options for the named filesystem.

mounts : A (space-separated) list of all clients currently mounting filesystems from this server.

nfs_threads : The maximal number of threads to be started by the nfs daemon.

Object: omd

Version: 1.4
Author: paul

Description:

This object starts the object manager daemon `omd`.

Object: oracle

Version: 1.10
Author: rs

Description:

This object starts the Oracle database.

Object: package

Version: 1.5
Author: dwb

Description:

This object checks the specified packages that have been installed on the machine and installs any that are missing. It also removes any that had been installed but have been removed from the list of required packages. The package directory can contain either executable files or subdirectories containing standard Sun packages. Packages are installed by executing any executable files and running `pkgadd` on the package(s) in any subdirectories. If any packages are actually installed, then a reboot is requested. If the installation of a package fails this will generate a warning but the installation of further packages will continue unless the `cautious` attribute is set to a non-null value. The `run` method performs the same actions as the `start` method, allowing new packages to be installed regularly from `cron`. The `verify` method allows the current package list to be checked. A check is made that each package on the list exists. If a specific package or space seperated list of packages is specified then a check is made that the specified package or packages exist. Only problems are reported unless "verify -v" is used. The file `/var/obj/conf/package.list` is used to maintain the list of installed packages. The file `/var/obj/conf/BACKOUT.package_name` is used to hold information on what sub-packages should be removed and in what order when a package is removed.

Resources:

list : A (space-separated) list of packages to install. The single value '`*`' indicates that all packages in the specified directory

should be installed.

packagedir : The directory containing the packages.

installed : List of packages already installed.

cautious : Non-null to stop further packages being installed after a package installation has failed.

tags : A space seperated list of extra tags which should be used when installing packages.

deftag : The default tag to use when installing packages. This is generated from the architecture of the machine that the package object is running on, e.g. m for sun4m, c for sun4c etc.

usetags : The list of tags mad by combining deftag and tags.

Object: patch

Version: 1.31
Author: paul
Also modified by: dwb,gdmr

Description:

This object checks the patches that have been applied to the machine and applies any that are missing. The patch directory can contain either executable files or subdirectories containing standard Sun patches, including an `installpatch` script. Patches are applied by executing any executable files and running `installpatch` in any subdirectories. If `use_fast` is set to a non-null value then the `fastpach` replacement for `installpatch` will be used to apply patches. If any patches are actually applied, then a reboot is requested. If the application of a patch fails this will generate a warning but the application of further patches will continue unless the `cautious` attribute is set to a non-null value. The `run` method performs the same actions as the `start` method, allowing new patches to be applied regularly from `cron`. The `verify` method allows the current patch list to be checked. A check is made that each patch on the list exists and is appropriate for the OS version. If a specific patch or space seperated list of patches is specified then a check is made that the specified patch or patches exist and are appropriate for the OS version. Only problems are reported unless "verify -v" is used. The file `/var/obj/conf/patch.list` is used to maintain the list of applied patches. If any of these patches are de-installed, then the patch must be removed from this file before `patch` will re-install it.

Resources:

list : A (space-separated) list of patches to apply. The single value '`*`' indicates that all patches in the specified directory should be applied.

installpatch_options : Options which should be passed to the installpatch script.

patchdir : The directory containing the patches.

applied : List of patches already applied.

use_fast : Non-null to enable use of the fastpatch replacement for the installpatch script. If fastpatch is used then `installpatch_options` should be appropriate to `fastpatch` rather than `installpatch`.

cautious : Non-null to stop further patches being applied after a patch application has failed.

Object: pcnfs

Version: 1.3
Author: dwb

Description:

This object starts the PCNFS daemon. The `pcnfsd_binary` resource can be used to alter which PCNFS daemon binary is used.

Object: plp

Version: 1.32
Author: paul
Last Modified by: jst
Also modified by: dwb

Description:

This object constructs all the necessary configuration files and starts the PLP print servers.

Resources:

printers : A (space-separated) list of printers served by this host. Strictly speaking, it's a list of spool directories rather than printers, so entries like `eucs/lp15` are allowed.

accountmail : A space-separated list of users who should receive the accounting report generated on this host.

accounthosts : If this is non-null, only hosts named in the list will be included in accounting reports.

Object: quotas

Version: 1.15
Author: dwb
Also modified by: gdmr

Description:

This is the quotas object, the `start` method determines whether quotas are already switched on and if they are not then it checks quotas on all partitions with quotas enabled and switches quotas on. The `run` method sets quotas for users with directories on partitions with quotas enabled from the NIS map quotas.byname. After the new quotas have been set a quota check is done. The `report` method generates a report of the quotas set on all partitions with quotas enabled. The `quota` method reports the quota set for a specified user or space seperated list of users. The `usage` method generates a list of users and their current disk usage sorted in order of decreasing disk usage for a specified partition (or specified partitions) or for all partitions with quotas enabled if none are specified. The `stop` method switches quotas off for all partitions with quotas enabled.

Resources:

quofsys : The list of filesystems with quotas enabled.

mail_report : If "yes" then a report of users who are over quota will be mailed to the addresses given by `report_to` and Cc'ed to the addresses given by `report_cc`.

report_to : The report of over quota users will be mailed to this address list.

report_cc : The report of over quota users will be Cc'd to this address list.

Object: reflector

Version: 1.4
Author: paul

Description:

This object controls the CU-SeeMe video-conferencing relector.

Resources:

motdfile : The name of a file containing the logon message for the reflector.

refmon : The reflector parameter REFMON.

confmgr : The reflector parameter CONF-MGR.

self_reflect : A non-null value sets the reflector parameter SELF_REFLECT.

conf_id : The reflector parameter CONF-ID.

cap : The reflector parameter CAP.

participants : The reflector parameter MAX-PARTICIPANTS.

senders : The reflector parameter MAX-SENDERS.

lurkers : The reflector parameter MAX-LURKERS.

admit_bcc : The reflector parameter ADMIT-GENERAL-BCC.

obtain_bcc : The reflector parameter OBTAIN-GENERAL-BCC.

unicast_ref : The reflector parameter UNICAST-REF.

nv_uc_port : The reflector parameter NV-UC-PORT.

vat_uc_port : The reflector parameter VAT-UC-PORT.

vat_conf_id : The reflector parameter VAT-CONF-ID.

mc_in : The reflector parameter MC-IN.

mc_out : The reflector parameter MC-OUT.

nv_mc_port : The reflector parameter NV-MC-PORT.

nv_mc_in : The reflector parameter NV-MC-IN.

nv_mc_out : The reflector parameter NV-MC-OUT.

vat_mc_port : The reflector parameter VAT-MC-PORT.

vat_mc_in : The reflector parameter VAT-MC-IN.

vat_mc_out : The reflector parameter VAT-MC-OUT.

Object: saf

Version: 1.3
Author: paul

Description:

This object configures the service access facility. At present, this includes only a very limited support for console logins.

Resources:

logins : A list of devices on which a login is to be allowed. Currently, only `console` is supported. In future, serial lines (eg. `ttya`) will also be supported.

prompt_*device* : The login prompt to display on the specified device.

term_*device* : The terminal type of the given device.

Object: samba

Version: 1.14
Author: ajs

Description:

This object starts the samba service. This provides SMB file and printing access for WindowsNT clients.

Resources:

printers : A list of printer tags to export.

type_*tag* : The type of printer for this tag, as known by WindowsNT.

options_*tag* : A list of options supported by the printer specified by this tag (eg 2up).

fsys : A list of filesystem tags to export, in addition to home directories.

path_*tag* : The pathname of the filesystem to export for this tag. The filesystem may be automounted from another server.

Object: snmp

Version: 1.11
Author: gdmr

Description:

This object starts the `SNMP` daemon.

Resources:

daemon : The version of the snmp daemon to run. sunv1, sunv2 and cmuv2 are the currently-supported versions.

The object makes use of the "location", "make", "model", "sno" and "hostid" info resources as well as the following which control what the corresponding SNMP queries return:

sysDescr :

sysLocation :

sysContact :

Object: squid

Version: 1.2
Author: jtb

Description:

This object runs the WWW Squid cache server.

Resources:

configfile : The pathname of the configuration file.

Object: ssh

Version: 1.8
Author: gdmr

Description:

This object starts the ssh daemon .

Resources:

ssh_daemon : The name of the daemon to run. The default is usually correct.

Object: syslog

Version: 1.31
Author: paul
Also modified by: ajs,dwb,gdmr

Description:

This object starts the `syslogd` daemon. The `prestart` method is used in the /etc/rc* startup files to start the syslog daemon before any of the services, particularly the DNS, starts up. It does a minimal start up with the pre-existing configuration file.

Resources:

configfile : The base `syslog.conf` configuration file. This file is passed through `m4` when `syslog` starts. A number of symbols are predefined, but these do not include the `LOGHOST` variable which is conventionally available. Direction of syslog messages to a remote log host can be achieved using the `additions` resource.

additions : A list of *tags* for additional lines to be added to `syslog.conf`.

add_*tag* : The additional line corresponding to *tag* to be added to the `syslog.conf` file.

Object: system

Version: 1.8
Author: gdmr
Last Modified by: dwb

Description:

This object manages /etc/system. Any changes from the existing version cause a reboot -r to take place.

Resources:

set : A list of variables to be set. Each entry requires an associated value, and optionally a tag and op.

default_set : Same as "set", but intended to define global defaults which should be applied to all systems.

tag_X : Optionally the name of the variable to be set. If not specified, default to X.

op_X : Optionally, the operation to be performed. Defaults to '='. Beware of shell meta-characters.

value_X :

quired. :

ndd : A list of protocol module variables to be set using "ndd". Each entry requires an associated driver and value, and optionally a tag. ..re driver_X The driver on which ndd should operate.

Object: teachclient

Version: 1.16
Author: dwb

Description:

This is the daily teaching client tidy up object, the `start` method starts up the screenblank daemon. The `run` method removes old non-root owned files from all directories listed in the `dirlist` resource. A check is also made that the screenblank daemon is still running.

Resources:

dirlist : A space seperated list of directories to be tidied up.

mail_report : If "yes" then a report of users who are over quota will be mailed to the addresses given by `report_to` and Cc'ed to the addresses given by `report_cc`.

report_to : The report of over quota users will be mailed to this address list.

report_cc : The report of over quota users will be Cc'd to this address list.

run_screenblank : If this resource is non-null then the screenblank daemon will be started.

cachedir : The name of the filesystem used by cachefs.

cachethresh : The percentage usage above which the cache directory usage will be reported.

acctdir : The name of the filesystem holding accounting info.

acctthresh : The percentage usage above which the accounting directory partition usage will be reported.

Object: update

Version: 1.36
Author: paul
Last Modified by: dwb

Description:

This object updates files in the root filesystem from the template containing the local modifications. If any files are actually changed, then a reboot is requested. Files with a name of the form `+DELETE:`*name* in the template cause the corresponding file *name* in the root to be deleted. Files with a name of the form `+HUSH:`*name* in the template inhibit the reboot request that would normally be generated when the corresponding file *name* is changed. The additional method `install` is used by the Sun auto-install `finish` script to install an initial version of the local files on the root of a new system, and by the `install add` method during installation of diskless clients.

Resources:

objlist : This is the list of objects to be copied into `/etc/obj` from `/usr/local/lcfg/obj`.

force : If this resource is null, then a timestamp file is checked and only files which have changed since the timestamp was updated are copied. If `force` is non-null, then the timestamp is ignored.

count : The number of files changed.

Object: updatelf

Version: 1.48
Author: paul
Last Modified by: dwb
Also modified by: jtb

Description:

This object updates the local filesystems. The additional method `monitor` is available to start the `lfu` monitor on any `updatelf` process which is already running (the `display` resource must be set). The `run` method accepts additonal arguments of the form '`-d` *directory*' or '`-p` *package-name*' to allow updating of a single package or directory.

Resources:

export : The source file system for the `lfu`.

disk : This resource combined with the `fs` resource specifies the destination file system(s) for the `lfu.`

fs : A (space-separated) list of local file systems for the `lfu`. (eg. `updatelf.fs local local1`).

fs_*filesystem* : (for each *filesystem* specified in the `fs` resource) A space-separated list of architectures for each local file system.

prog : The skeleton `lfu` program which is used to generate `progfile` when the object starts.

progfile : The generated program file used by `lfu`.

netgroups : A (space-separated) list of netgroups in the order that they are to be handled in the program.

packs_*name* : Any resources of this form are used by `gengrp` to generate netgroups. All the packages specified in the resource are used to create a netgroup with the name `PACKS_`*`hostname_name`*. This provides a more convenient way of declaring netgroups which are primarily used by just a single machine.

action_*netgroup* : (for each *netgroup* in the `netgroups` resource) The action for each netgroup. This must be `ignore`, `link`, `copy`, or `cache`.

linkto_*netgroup* : (for each *netgroup* in the `netgroups` resource) The cluster to use if the action for this netgroup is `link` or `cache`.

notify : The mail address for notification of errors.

lfuversion : The version of `lfu` to use.

rootfiles : A file containing the list of files which are to be changed to root-owned on copying.

bootfiles : A file containing the list of files which require the machine to be rebooted when they change. They have the action `keep`.

keepfiles : A file containing the list of files which require users to log out and in again if they change. They have the action `keep` assigned in the program.

nightly_log : The logfile for the (usually) nightly logs.

logall : Any file satisfying this condition will have action `logall`. (any examination of the file is logged regardless of changes).

lfu_options : A list of flags that are passed to the `lfu` command.

df_thresh : The disk usage threshold (as a percentage) for reporting that the destination file system is getting full.

display : The display variable for use with xlfumon.

debug : Non-null to enable debugging of `updatelf` script (see also `lfu_options`).

Object: volmgt

Version: 1.11
Author: paul
Last Modified by: gdmr

Description:

This objects constructs all the necessary configuration files and starts the volume manager.

Resources:

configfile : The `vold.conf` configuration file used by vold.

Object: www

Version: 1.36
Author: jtb
Last Modified by: rs
Also modified by: paul,root,dwb,iro

Description:

This is a WWW server object. It starts the server, runs scripts which set up the links to staff and student pages, and to information about packages availables on the DCS system. It also cycles the logs for the www object, www access, www cache and www proxy.

Resources:

server : The pathname of the server to run.

Object: xconfig

Version: 1.3
Author: gdmr

Description:

This object writes out a /etc/XDisplayOptions file, which is later used by the x11 startup script to configure special display options (such as multi-headed positioning).

Object: xdm

Version: 1.22
Author: gdmr
Last Modified by: dwb
Also modified by: paul

Description:

This object starts the `xdm` daemon. (The font-server is started from its own object.)

Resources:

servers : This resources is passed as the `server` parameter to `xdm`. If the special value `local` is specified, then `xdm` will manage a server on the local display only if there is no console login running. The default values allow a console login or xdm to be selected simply be enabling or disabling the console logins with the `saf` object.

local : The value to be passed to `xdm` as the `servers` argument when `local` is specified for the `servers` resource.

config : If this resource is present, the specified file is passed through `cpp` and then used as the configuration file for `xdm`. The preprocessor symbol `DEF_CONFIG` is defined with the name of the default config file so that this can be included if required.

X_version : The version of X11.

port : The port number to listen for requests.

access_direct :

access_indirect :

access_chooser_list : These resources control the generated access file. See the xdm man-page for details.

Object: xfs

Version: 1.6
Author: gdmr

Description:

This object starts the `font`server daemon. (`xdm` is started from its own object.)

Object: xntpd

Version: 1.27
Author: paul
Last Modified by: gdmr

Description:

This object constructs all the necessary configuration files and starts the `xntpd` time daemon. The defaults are appropriate for leaf nodes, while the NTP_SERVER macro is normally used to set suitable values on machines which are expected to supply time to others. There should be at least one NTP_SERVER on each subnet. Machines on wire CS-A (129.215.160) should be explicitly synchronised; broadcasts are not sent to this wire, as a workaround for a problem with explicit servers on a wire to which a machine is broadcasting.

Resources:

servers : A (space-separated) list of NTP version 3 servers.

v2servers : A (space-separated) list of NTP version 2 servers.

peers : A (space-separated) list of NTP version 3 peers.

v2peers : A (space-separated) list of NTP version 2 peers.

ntpdate_fallback : A (space-seprarated) list of fallback servers which should be used to synchronise to should there be no configured servers or peers.

broadcast : If any servers or peers are defined, broadcast time to any attached subnets. The default is "yes".

configfile : The configuration file.

driftfile : The drift file.

Object: xtpp

Version: 1.11
Author: morna
Last Modified by: dwb

Description:

This is the xtpp object, it manages the xtpp mail system. The `install` method can be used to install the xtpp mail system on a new mail server. The `link` method can be used to add links to the xtpp libraries on another host. The `setlogs` method performs the daily tidying up of the xtpp specific log files.

Resources:

qmgr_maxchans : Sets the maximum number of runable channels for the queue manager on start up.

Object: yp

Version: 1.13
Author: ajs
Also modified by: gdmr

Description:

This object configures the YP. It doesn't directly control any daemons as these have to be started before any of the "lcfg" objects. It transfers YP maps to make a machine a slave server and manipulates the YP bindings so a machine can be bound to a particular server.

Resources:

type : The type of YP service on this machine. Valid options are `client` and `slave`.

servers : A list of servers to `ypbind` to. Specifying no servers forces `ypbind` to broadcast for it's YP maps. `master`W The YP master from which the YP maps are sucked.

secure_netmask : The netmask which will be applied to entries in the securenets file.

extra_nets : Networks which will be added to the securenets file in addition to those implicitly added by virtue of there being interfaces on the machine attached to them.