Sun Management Center Change Manager 1.0.1 Reference Manual

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 817–0892–10
May 2003

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REPONDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.

Adobe PostScript™

030430@5533

Contents

Preface

This reference manual contains the man pages for the Sun™ Management Center Change Manager product, henceforth referred to as Change Manager.

Who Should Use This Book

This book is intended for anyone responsible for performing one or more of these Change Manager operations:

- Installing and configuring the Change Manager software on the Change Manager server
- Managing deployment objects and audit objects in the Change Manager repository
- Managing hosts on the Change Manager server
- Creating the Solaris™ Flash archives for use with Change Manager
- Deploying software stacks to managed hosts
- Auditing software on managed hosts

Related Books

- *Sun Management Center Change Manager 1.0.1 Release Notes*
- *Sun Management Center Change Manager 1.0.1 Administration Guide*
- *Solaris 9 Installation Guide*
- *Sun Management Center 3.5 Installation and Configuration Guide*
- *Sun Management Center 3.5 User's Guide*

Accessing Sun Documentation Online

The docs.sun.com^{SM} Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is `http://docs.sun.com`.

Ordering Sun Documentation

Sun Microsystems offers select product documentation in print form. For a list of documents and how to order them, see ‘‘Buy printed documentation’’ at `http://docs.sun.com`.

Change Manager System Administration Commands

NAME

bart – basic audit reporting tool

SYNOPSIS

```
/usr/bin/bart create [ -n ] [ -R root_directory ] [ -r rules_file | - ]
```

```
/usr/bin/bart create [ -n ] [ -R root_directory ] -I [ file_name ...]
```

```
/usr/bin/bart compare [ -i attribute ] [ -p ] [ -r rules_file | -
      ] control-manifest test-manifest
```

DESCRIPTION

The `bart`(1MCM) command is a tool that performs a file-level check of the software contents of a system. Users can optionally specify the files to track and the types of discrepancies to flag by means of a rules file. See `bart_rules`(4CM).

The `bart` command performs two basic functions:

`bart create` The manifest generator tool takes a file-level ‘‘snapshot'' of a system. The output is a catalog of file attributes referred to as a ‘‘manifest.'' See `bart_manifest`(4CM).

Users can specify the list of files to be cataloged in three ways. Use `bart create` with no options, specify the files by name on the command line, or create a rules file with directives that specify which the files to monitor. See `bart_rules`(4CM).

By default, the manifest generator catalogs all attributes of all files in the root (`/`) file system. File systems mounted on the root file system are cataloged only if they are of the same type as the root file system.

For example, `/`, `/usr`, and `/opt` are separate UFS file systems. `/usr` and `/opt` are mounted on `/`. Therefore, all three file systems are cataloged. However, `/tmp`, also mounted on `/`, is not cataloged because it is a TMPFS file system. Mounted CD-ROMs are not cataloged since they are HSFS file systems.

`bart compare` The report tool compares two manifests. The output is a list of per-file attribute discrepancies. These discrepancies are the differences between two manifests: a control manifest and a test manifest. A discrepancy is a change to any attribute for a given file cataloged by both manifests. Note that a new file or a deleted file in a manifest is reported as a discrepancy.

The reporting mechanism provides two types of output: verbose and programmatic. Verbose output is localized and presented on multiple lines, while programmatic output is more easily parsable by other programs. See OUTPUT.

By default, the report tool generates verbose output where all discrepancies are reported except for modified directory timestamps (the `dirmtime` attribute).

Note – To ensure consistent and accurate comparison results, both *control-manifest* and *test-manifest* must be built with the same rules file.

Use the rules file to ignore specified files or subtrees when you generate a manifest or compare two manifests. Users can compare manifests from different perspectives by re-running the `bart compare` command with different rules files.

OPTIONS

The following options are supported:

`-i` *attribute* ... Specifies the file attributes to be ignored globally. This option produces the same behavior as supplying the file attributes to a global `IGNORE` keyword in the rules file. See `bart_rules`(4CM).

`-I` [*file_name*...] Inputs list of files. The file list can be specified at the command line or read from standard input.

`-n` Prevents computation of content signatures for all regular files in the file list.

`-p` Displays manifest comparison output in ''programmatic mode,'' which is suitable for programmatic parsing. The output is not localized.

`-r` *rules_file* Uses *rules_file* to specify which files and directories to catalog, and to define which file attribute discrepancies to flag. If *rules_file* is `-`, then the rules are read from standard input. See `bart_rules`(4CM) for the definition of the syntax.

`-R` *root_directory* Specifies the root directory for the manifest. All paths specified by the rules, and all paths reported in the manifest, are relative to *root_directory*.

OPERANDS

The following operands are supported:

control-manifest Manifest created by `bart create` on the control system.

test-manifest Manifest created by `bart create` on the test system.

OUTPUT

The `bart create` and `bart compare` commands write output to standard output, and write error messages to standard error.

The `bart create` command generates a system manifest. See `bart_manifest`(4CM).

When the `bart compare` command compares two system manifests, it generates a list of file differences. By default, the comparison output is localized. However, if the `-p` option is specified, the output is generated in a form that is suitable for programmatic manipulation.

Default Format

```
filename
    attribute    control:xxxx test:yyyy
```

filename

Name of the file that differs between *control-manifest* and *test-manifest*. For file names that contain embedded whitespace or newline characters, see Quoting Syntax in `bart_manifest`(4CM).

attribute

The name of the file attribute that differs between the manifests that are compared. *xxxx* is the attribute value from *control-manifest*, and *yyyy* is the attribute value from *test-manifest*. When discrepancies for multiple attributes occur for the same file, each difference is noted on a separate line.

The following default output shows the attribute differences for the `/etc/passwd` file. The output indicates that the `size`, `mtime`, and `contents` attributes have changed.

```
/etc/passwd:
  size  control:74  test:81
  mtime  control:3c165879  test:3c165979
  contents  control:daca28ae0de97afd7a6b91fde8d57afa
    test:84b2b32c4165887355317207b48a6ec7
```

Programmatic Format

filename attribute control-val test-val [*attribute control-val test-val*] *

filename

Same as *filename* in the default format.

attribute control-val test-val

A description of the file attributes that differ between the control and test manifests for each file. Each entry includes the attribute value from each manifest. See `bart_manifest`(4CM) for the definition of the attributes.

Each line of the programmatic output describes all attribute differences for a single file.

The following programmatic output shows the attribute differences for the `/etc/passwd` file. The output indicates that the `size`, `mtime`, and `contents` attributes have changed.

```
/etc/passwd size 74 81 mtime 3c165879 3c165979
    contents daca28ae0de97afd7a6b91fde8d57afa 84b2b32c4165887355317207b48a6ec7
```

Manifest Generator

`0`	Success
`1`	Non-fatal error when processing files; for example, permission problems
`>1`	Fatal error; for example, invalid command-line options

Report Tool

`0`	No discrepancies reported
`1`	Discrepancies found
`>1`	Fatal error executing comparison

EXAMPLES

EXAMPLE 1 Creating a Default Manifest Without Computing Checksums

The following command line creates a default manifest, which consists of all files in the / file system. The `-n` option prevents computation of checksums, which causes the manifest to be generated more quickly.

```
bart create -n
```

EXAMPLE 2 Creating a Manifest for a Specified Subtree

The following command line creates a manifest that contains all files in the `/home/nickiso` subtree.

```
bart create -R /home/nickiso
```

EXAMPLE 3 Creating a Manifest by Using Standard Input

The following command line uses output from the `find`(1) command to generate the list of files to be cataloged. The `find` output is used as input to the `bart create` command that specifies the `-I` option.

```
find /home/nickiso -print | bart create -I
```

EXAMPLE 4 Creating a Manifest by Using a Rules File

The following command line uses a rules file, `rules`, to specify the files to be cataloged.

```
bart create -r rules
```

EXAMPLE 5 Comparing Two Manifests and Generating Programmatic Output

The following command line compares two manifests and produces output suitable for parsing by a program.

```
bart compare -p manifest1 manifest2
```

EXAMPLE 6 Comparing Two Manifests and Specifying Attributes to Ignore

The following command line compares two manifests. The `dirmtime`, `lnmtime`, and `mtime` attributes are not compared.

```
bart compare -i dirmtime lnmtime mtime manifest1 manifest2
```

EXAMPLE 7 Comparing Two Manifests by Using a Rules File

The following command line uses a rules file, `rules`, to compare two manifests.

```
bart compare -r rules manifest1 manifest2
```

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWbart
Interface Stability	Evolving

SEE ALSO

`find`(1), `bart_manifest`(4CM), `bart_rules`(4CM), `attributes`(5)

NAME

changemgr – Sun Management Center Change Manager command-line interface

SYNOPSIS

`/opt/SUNWichange/bin/changemgr` *command* [*options*] [*operands*]

DESCRIPTION

The `changemgr`(1MCM) command is the command-line interface for the Sun Management Center Change Manager, henceforth referred to as Change Manager. This command-line interface performs the same operations that can be performed by using the browser user interface, such as software deployment tasks and system audit tasks.

Change Manager commands must be run by an authenticated user.

The command-line interface can be used to initiate a Change Manager session. A Change Manager session is a subshell in which you can run Change Manager commands as an authenticated user. You authenticate when you initiate the session. All operations run within the session are owned by the authenticated user.

The command-line interface can also run custom scripts that execute multiple Change Manager commands. The script support facilitates the execution of multiple Change Manager operations. Authentication is performed once for the script instead of once per command-line invocation.

OPTIONS

The `changemgr` command supports several command-line options.

Authentication Options

Other than the `changemgr help` commands, all commands must be authenticated. In the context of a session, the session's authenticated identity is used.

The following authentication options are supported:

`-p` *file*

file consists of a single line, which contains the password. If *file* is `-`, then the user can supply the password as standard input.

If the `-p` option is not supplied, then the `changemgr` command prompts the user for his password.

`-u` *username*

Specifies the user name to authenticate. If the `-u` option is not supplied, the user is the current UNIX real user ID, as reported by `id`(1M).

Common Options

These options are used by more than one command:

`-d` *domain*

Specifies the Sun Management Center administrative domain on which to operate. In the context of a session, the default is the domain specified by the session, if any. By default, *domain* is the user's home domain.

`-o` *format*

format is a blank-separated list or comma-separated list of property names. If you separate the property names with spaces, make sure that you surround the list of property names with quotes. The specified property values are displayed in a *name*=*value* format. If *format* is specified as `all`, then all properties are displayed. The output is suitable for programmatic parsing.

The output lists each file or folder on a line by itself. The name can be followed by property lines, which consist of a tab, property name, equals sign, and a property value. Each file or folder entry is separated from the next entry by a blank line.

For example, the output is arranged as follows:

```
path
        name=value
        . . .

path
        name=value
        . . .

. . .
```

OPERANDS

The following operands are supported:

filepath — An absolute path to a file or a relative (to the current directory) path to a file. This file path is not in the Change Manager repository.

relpath — Path to a file-like object (including a folder) that is relative to the top of the Change Manager repository.

relfilepath — Path to a file-like object (*not* including a folder) that is relative to the top of the Change Manager repository.

reldirpath — Path to a folder-like object that is relative to the top of the Change Manager repository.

.*type* — File name suffix that specifies the file type. File type suffixes are: `.flar` for archives, `.miniroot` for boot images, `.bmft` for manifests, `.brul` for audit rules files, `.txt` for reports, and `.cmsp` for shared profiles. Folders do not require a file name suffix.

topopath — Path to a topology object (including a host group) that is relative to the top of the selected administrative domain.

hostpath — Path to a managed host that is relative to the top of the selected administrative domain.

hostname — Network name of a host, for example, `netherfield.sun.com`.

grouppath — Path to a host group that is relative to the top of the selected administrative domain.

SUBCOMMANDS

The following sections describe the `changemgr` subcommands.

Sessions

`changemgr session` [`-u` *username*] [`-p` *file*] [`-d` *domain*] [*command* [*command-arguments*]]

Run the specified *command* in the context of a Change Manager session so that individual commands in a script (*command*) do not need authentication and startup overhead. The authentication and startup overhead is amoritized over all of the commands.

command is normally an `sh`(1) or `ksh`(1) script that contains Change Manager commands in the form of the command-line interface.

If *command* is `sh` or `ksh`, a subshell is spawned to create an interactive session. You are required to authenticate to initiate the session.

If *command* is not supplied, then an interactive subshell of `$SHELL` starts, if known to be compatible. If `$SHELL` is not compatible, then an interactive `ksh` subshell starts.

Note – The `csh`(1) shell cannot be used to run scripts or initiate a session.

File Management Operations

`changemgr mkdir` [`-u` *username*] [`-p` *file*] *new_reldirpath*...

Create one or more folders in the Change Manager repository.

`changemgr import` [`-u` *username*] [`-p` *file*] *filepath*.[*type*] *relfilepath* . *type*

Import a single file, *filepath*.[*type*], to the repository as *relfilepath* . *type*. The file being imported can have any file suffix, but the file name in the repository *must* have the appropriate suffix.

`changemgr import` [`-u` *username*] [`-p` *file*] *filepath* . *type*... *reldirpath*

Import one or more files to the specified folder, *reldirpath*, in the repository.

Because this command uses the original file names when creating the files in the repository, the original names *must* have the appropriate suffixes.

`changemgr export` [`-u` *username*] [`-p` *file*] *relfilepath filepath*

Export a single file, *relfilepath*, from the repository as *filepath*.

`changemgr export` [`-u` *username*] [`-p` *file*] *relfilepath*... *dirpath*

Export one or more files to the specified folder, *dirpath*, outside of the repository.

`changemgr files` [`-u` *username*] [`-p` *file*] [`-l`] [`-d`] [`-R`] [`-o` *format*] [*relpath*...]

List the specified files and folders, or the contents of the specified folders. When no path is specified, the objects in the root of the repository are listed.

The default output format is one file or folder name per line.

`-d` Presents information about the folder itself, rather than about the folder's contents.

`-l` Presents more information in tabular output. This output is not suitable for programmatic parsing.

`-R` Recursively lists the contents of a folder.

`changemgr delete` [`-u` *username*] [`-p` *file*] *relpath*...
Delete the specified files and folders.

Note that only empty folders can be deleted.

`changemgr fileset` [`-u` *username*] [`-p` *file*] [`-s` *name=value*]... [`-s` *name*]... *relpath*...
Set properties for the specified files and folders by using the `-s` *name=value* option. The `-s` option with just the property name deletes the property.

`-s` *name=value* Specifies one or more *name=value* pairs. *name* is the property name, and *value* is the property value. Supply the `-s` option for each property value you want to set. If *value* is blank, then the property is assigned an empty value.

`-s` *name* Specifies one or more property names to delete, where *name* is the property name. Supply the `-s` option for each property you want to delete.

`changemgr filemove` [`-u` *username*] [`-p` *file*] *old_relpath*... *new_dirpath*
Move files and folders to another folder. The original file and folder names are unchanged. The destination folder *must* already exist.

old_relpath can be a folder or a file.

`changemgr filemove` [`-u` *username*] [`-p` *file*] *old_relpath*`.`*type* *new_relpath*`.`*type*
Rename a file or a folder. The type of the renamed file *must* stay the same.

Topology Operations

`changemgr mkgroup` [`-u` *username*] [`-p` *file*] [`-d` *domain*] *new_grouppath*...
Create one or more host groups.

`changemgr hosts` [`-u` *username*] [`-p` *file*] [`-l`] [`-g`] [`-R`] [`-d` *domain*] [`-o` *format*] [*topopath*...]
List information about *topopath*, which represents the specified managed hosts or host groups. With no path arguments, information is listed about the managed hosts and host groups in the root of the administrative domain.

The default output format is one managed host or host group name per line.

`-g` Presents information about the group itself, rather than about the group's contents.

`-l` Presents more information in tabular output. This output is not suitable for programmatic parsing.

`-R` Recursively lists the contents of a group.

`changemgr add` [`-u` *username*] [`-p` *file*] [`-d` *domain*] *hostname hostpath*
Register a network host name as a Sun Management Center host name. The host path includes the host group and the host name. The name of the managed host can be different from the network host name.

`changemgr add` [`-u` *username*] [`-p` *file*] [`-d` *domain*] *hostname*... *grouppath*
Add the specified managed hosts to the specified host group, with the managed host names equal to the network host names.

`changemgr remove` [`-u` *username*] [`-p` *file*] [`-d` *domain*] *topopath...*
Remove managed hosts and host groups from the topology.

`changemgr hostset` [`-u` *username*] [`-p` *file*] [`-d` *domain*] [`-s` *name=value*]... [`-s` *name*]... *topopath...*
Set properties for the specified managed hosts and host groups by using the `-s` *name=value* option. The `-s` option with just the property name deletes the property.

`-s` *name=value* Specifies one or more *name=value* pairs. *name* is the property name, and *value* is the property value. Supply the `-s` option for each property value you want to set. If *value* is blank, then the property is assigned an empty value.

`-s` *name* Specifies one or more property names to delete, where *name* is the property name. Supply the `-s` option for each property you want to delete.

`changemgr hostmove` [`-u` *username*] [`-p` *file*] [`-d` *domain*] *old_topopath... new_grouppath*
Move managed hosts or host groups to another host group. The destination host group *must* already exist.

`changemgr hostmove` [`-u` *username*] [`-p` *file*] [`-d` *domain*] *old_topopath new_topopath*
Rename a single managed host or host group.

Host Operations

`changemgr update` [`-u` *username*] [`-p` *file*] [`-d` *domain*] [`-x` *operation*] *topopath...*
Update the specified managed hosts to conform to the configuration specified by their properties.

If *topopath* is a host group, all members of the host group are updated.

`-x` *operation* Specifies the action to take after the update completes. If *operation* is `reboot`, then activate the newly installed software stack and reboot. If *operation* is `halt`, then activate the newly installed software stack and halt. The default operation is to reboot the managed host.

`changemgr fallback` [`-u` *username*] [`-p` *file*] [`-d` *domain*] [`-x` *operation*] *topopath...*
Restore the specified managed hosts to their state prior to the last `changemgr update` operation. This action only undoes the last update operation. This action does not change the parameters associated with the managed host. After the fallback operation, the managed host's running configuration will not match the parameters selected for it, which is the case immediately prior the update operation.

If *topopath* is a host group, all members of the host group are restored.

`-x` *operation* Specifies the action to take after the fallback completes. If *operation* is `reboot`, then activate the newly selected software stack and reboot. If *operation* is `halt`, then activate the newly selected software stack and halt. The default operation is to reboot the managed host.

`changemgr reinstall` [`-u` *username*] [`-p` *file*] [`-d` *domain*] *topopath*...

Reinstall the specified managed hosts. The reinstallation is equivalent to this:

```
# reboot -- net - install
```

If *topopath* is a host group, all members of the host group are reinstalled.

`changemgr setup` [`-u` *username*] [`-p` *file*] [`-d` *domain*] *topopath*...

Set up files for initial installation. This operation is required before manually running `boot net - install` on the consoles of managed hosts.

If *topopath* is a host group, all files for the members of the host group are set up.

`changemgr reboot` [`-u` *username*] [`-p` *file*] [`-d` *domain*] *topopath*...

Reboot the specified managed hosts.

If *topopath* is a host group, all members of the host group are rebooted.

`changemgr halt` [`-u` *username*] [`-p` *file*] [`-d` *domain*] *topopath*...

Halt the specified managed hosts.

If *topopath* is a host group, all members of the host group are halted.

`changemgr manifest` [`-u` *username*] [`-p` *file*] [`-d` *domain*] `-o` *relpathprefix* [`-r` *relfilepath*`.brul`] *topopath*...

Create manifests for the specified managed hosts.

`-o` *relpathprefix* — Specifies the prefix to use when creating manifests. The host name and suffix are appended to the prefix to form the name of the manifest.

`-r` *relfilepath*`.brul` — Specifies the audit rules file to use when building manifests.

`changemgr audit` [`-u` *username*] [`-p` *file*] [`-d` *domain*] `-o` *relfilepath*`.txt` [`-r` *relfilepath*`.brul`] *relfilepath*`.bmft` *topopath*...

Compare managed host contents against a baseline manifest.

`-o` *relfilepath*`.txt` — Specifies the file path of the report.

`-r` *relfilepath*`.brul` — Specifies the audit rules file to use when auditing hosts.

`changemgr info` [`-u` *username*] [`-p` *file*] [`-d` *domain*] `-o` *relfilepath*`.txt` *topopath*...

Get software status information about the specified managed hosts. Store the results in the specified report.

`-o` *relfilepath*`.txt` — Specifies the file path of the report.

Job Management Operations

`changemgr jobs` [`-u` *username*] [`-p` *file*] [`-l`] [`-o` *format*] [*id*...]

Display the status of all outstanding jobs or of specified jobs.

`-l` — Presents more information in tabular output. This output is not suitable for programmatic parsing.

`changemgr kill` [`-u` *username*] [`-p` *file*] *id*...
Cancel currently running jobs or pending jobs.

`changemgr ack` [`-u` *username*] [`-p` *file*] *id*...
Acknowledge the completion of the specified jobs. This action discards the status of the specified jobs.

Use this command to purge completed job entries from the job list that were initiated by the browser interface.

Miscellaneous

`changemgr help`
Provide a one-line summary of the available subcommands. No authentication is required.

`changemgr help` *subcommand*
Provide a summary of the specified subcommand. No authentication is required.

SEE ALSO

`csh`(1), `date`(1), `ksh`(1), `sh`(1), `id`(1M)

EXAMPLES

EXAMPLE 1 Running Commands in an Interactive Change Manager Session

The following example shows an interactive Change Manager session. The `changemgr session` command starts a subshell in which you can run authenticated `changemgr` commands.

This example shows how to purge a completed job from the job queue. This job, `IC_1`, was initiated from the browser interface. When the tasks are completed, exit the session by typing `exit` at the subshell prompt.

```
$ changemgr session
Password: password
$ changemgr jobs -l IC_1
IC_1     succeeded
$ changemgr ack IC_1
$ changemgr jobs l IC_1
$ exit
```

EXAMPLE 2 Running Scripts in a Change Manager Session

This example shows how to use the `changemgr session` command to run a script.

The following command line runs the `deploy-web` script.

```
$ changemgr session deploy-web web.flar host1
```

The `deploy-web` script contains the following:

```
$ cat deploy-web
#/bin/sh
changemgr import "$1" /
changemgr fileset -s MediaName=s9.miniroot "$1"
changemgr hostset -s base_config_flar_archive="/$1" "$2"
changemgr update "$2"
$
```

NAME

cmgetprop – Get Change Manager property value

SYNOPSIS

cmgetprop *property-name*

DESCRIPTION

The cmgetprop(1MCM) command writes the value of the specified property to standard output. Note that no value is returned when the property is not set.

Use the cmgetprop command in deployment finish scripts to get property values. Change Manager finish scripts are stored in the /etc/ichange.d directory.

The cmgetprop command is included in the $PATH supplied to the Change Manager finish scripts.

EXAMPLES

EXAMPLE 1 Using the cmgetprop Command

The following line might exist in a Change Manager finish script.

```
FILENAME=‘cmgetprop FNAME‘
```

This code assigns the value of the FNAME property to the FILENAME variable.

SEE ALSO

Sun Management Center Change Manager 1.0.1 Administration Guide

NAME

flar – administer flash archives

SYNOPSIS

flar `create -n` *name* `[-R` *root*`] [-H] [-I] [-S] [-c] [-t [-p` *posn*`] [-b` *blocksize*`]] [-i` *date*`] [-u` *section...*`] [-U` *key=value...*`] [-m` *master*`] [-f [`*filelist* `| -] [-F]] [-a` *author*`] [-e` *descr* `| -E` *descr_file*`] [-T` *type*`] [-x` *exclude...*`] [-y` *include...*`] [-z` *filelist...*`] [-X` *filelist...*`]` *archive*

flar `combine [-d` *dir*`] [-u` *section...*`] [-t [-p` *posn*`] [-b` *blocksize*`]]` *archive*

flar `split [-d` *dir*`] [-u` *section...*`] [-f] [-S` *section*`] [-t [-p` *posn*`] [-b` *blocksize*`]]` *archive*

flar `info [-l] [-k` *keyword*`] [-t [-p` *posn*`] [-b` *blocksize*`]]` *archive*

DESCRIPTION

The `flar` command is used to administer flash archives. A flash archive is an easily transportable version of a reference configuration of the Solaris operating environment, plus other optional software. Such an archive is used for the rapid installation of the Solaris software on large numbers of machines. You can create a flash archive using either `flar` with the `create` subcommand or the `flarcreate`(1MCM) command. See `flash_archive`(4CM).

In flash terminology, a system on which an archive is created is called a *master*. The system image stored in the archive is deployed to systems that are called *clones*.

An archive is created with the `create` subcommand. It contains all the files that are in a system image.

You can run `flar create` in multiuser or single-user mode. You can also use the command when the master system is booted from the first Solaris software CD or from a Solaris net image. Archive creation should be performed when the master system is in as stable a state as possible.

Following the creation of a flash archive, you can use custom JumpStart to clone the archive on multiple systems.

SUBCOMMANDS

The `flar` command includes subcommands for creating, combining, splitting, and providing information about archives. A subcommand is the first argument in a `flar` command line. The subcommands are as follows:

`create`

Create a new flash archive, of a name you specify with the `-n` argument, based on the currently running system.

The `create` subcommand requires superuser privileges.

`combine`

Combine the individual sections that make up an archive into the archive. If *dir* is specified (see `-d` option), the sections will be gathered from *dir*. Otherwise, they will be gathered from the current directory. Each section is assumed to be in a separate file, the names of which are the section names. At a minimum, the archive cookie (`cookie`), archive identification (`identification`), and archive files (`archive`) sections must be present. If `archive` is a directory, its contents are archived using `cpio` prior to inclusion in the archive. If so specified in the `identification` section, the contents are compressed.

Note that no validation is performed on any of the sections. In particular, no fields in the `identification` section are validated or updated. See `flash_archive`(4CM) for a description of the archive sections.

`info`
: Extract information on an archive. This subcommand is analogous to `pkginfo`.

`split`
: Split an archive into one file for each section of the archive. Each section is copied into a separate file in *dir*, if *dir* is specified (see `-d` option), or the current directory if it is not. The files resulting from the split are named after the sections. The archive cookie is stored in a file named `cookie`. If *section* is specified (see `-u` option), only the named section is copied.

The options for each subcommand are described in OPTIONS.

OPTIONS

The `create` subcommand has one required option:

`-n` *name*
: Is the value of the `content_name` keyword. See `flash_archive`(4CM).

Following are the options for the `create` subcommand. Many of these options supply values for keywords in the identification section of a file containing a flash archive. See `flash_archive`(4CM) for a description of these keywords.

`-a` *author*
: Provides an author name for the archive identification section of the new flash archive. If you do not specify `-a`, no author name is included in the identification section.

`-c`
: Compresses the archive using `compress`(1).

`-e` *descr*
: Is the description to be included in the archive as the value of the `content_description` archive identification key. This option is incompatible with `-E`.

`-E` *descr_file*
: Is the description to be used as the value of the archive identification `content_description` key as retrieved from the file *descr_file*. This option is incompatible with `-e`.

`-f` *filelist*
Uses the contents of *filelist* as a list of files to include in the archive. The files are included in addition to the usual file list, unless `-F` is specified (see the `-F` option). If *filelist* is `-`, the list is taken from standard input.

`-F`
Includes only files in the list specified by `-f`. This option makes `-f` *filelist* an absolute list, rather than a list that is appended to the usual file list.

`-H`
Does not generate a hash identifier.

`-i` *date*
By default, the value for the `creation_date` field in the identification section is generated automatically, based on the current system time and date. If you specify the `-i` option, *date* is used instead.

`-I`
Ignores integrity check. To prevent you from excluding important system files from an archive, `flar` runs an integrity check. This check examines all files registered in a system package database and stops archive creation if any of them are excluded. Use this option to override this integrity check.

`-m` *master*
By default, the value for the `creation_master` field in the identification section is the name of the system on which you run `flar create`, as reported by `uname -n`. If you specify `-m`, *master* is used instead.

`-R` *root*
Creates the archive from the file system tree mounted at *root*. If you do not specify this option, `flar` creates an archive from a file system mounted at `/`.

`-S`
Skips the disk space check. Without `-S`, `flar` builds a compressed archive in memory before writing the archive to disk, to ensure you have sufficient disk space. Use `-S` to skip this step. The result of the use of `-S` is a significant decrease in the time it takes to create an archive.

`-T` *type*
Is the content type included in the archive as the value of the `content_type` archive identification key. If you do not specify `-T`, the `content_type` keyword is not included.

`-u` *section ...*
Include the user-defined section located in the file *section* in the archive. *section* must be a blank-separated list of section names as described in the `flash_archive`(4CM) man page.

`-U` *key=value...*
Includes one or more user-defined keywords and their values in the archive identification section. See `flash_archive`(4CM).

`-x` *exclude* ...

Excludes the file or directory *exclude* from the archive. Note that the *exclude* file or directory is assumed to be relative to the alternate *root* specified using `-R`. If the parent directory of the file *exclude* is included with the `-y` option (see `-y` *include*), then only the specific file or directory specified by *exclude* is excluded. Conversely, if the parent directory of an included file is specified for exclusion, then only the file *include* is included. For example, if you specify:

```
-x /a -y /a/b
```

all of `/a` except for `/a/b` is excluded. If you specify:

```
-y /a -x /a/b
```

all of `/a` except for `/a/b` is included.

`-X` *filelist* ...

Uses the contents of *filelist* as a list of files to exclude from the archive. If *filelist* is –, the list is taken from standard input.

`-y` *include* ...

Includes the file or directory *include* in the archive. Note that the *exclude* file or directory is assumed to be relative to the alternate *root* specified using `-R`. See the description of the `-x` option for a description of the interaction of the `-x` and `-y` options.

`-z` *filelist* ...

Is a list of files prefixed with a plus (+) or minus (`-`). A plus indicates that a file should be included in the archive. The minus indicates exclusion. If *filelist* is –, the list is taken from standard input.

The options for the `info` subcommand are as follows:

`-k` *keyword*

Returns only the value of the keyword *keyword*.

`-l`

Lists all files in the archive. Does not process content from any sections other than the `archive` section.

Following are the `info` options to use with tape archives:

`-b` *blocksize*

Is the block size to use when creating the archive. If not specified, a default block size of 64K is used.

`-p` *posn*

Specifies the position on the tape device where the archive should be created. If not specified, the current position of the tape device is examined.

`-t`

Indicates that the archive to be analyzed is located on a tape device. The path to the device is specified by *archive* (see OPERANDS).

The options for the `split` and `combine` (split and combine archives) subcommands are as follows:

`-d` *dir* Retrieves sections from *dir*, rather than from the current directory.

`-f` (Used with `split` only.) Extracts the `archive` section into a directory called `archive`, rather than placing it in a file of the same name as the section.

`-S` *section* (Used with `split` only.) Extracts only the section named *section* from the archive.

`-u` *section...* Appends *section* to the list of sections to be included. The default list includes the `cookie`, `identification`, and `archive` sections. *section* can be a single section name or a space-separated list of section names.

The following options are used with tape archives (with both `split` and `combine`):

`-b` *blocksize* Is the block size to be used when creating the archive. If not specified, a default block size of 64K is used.

`-p` *posn* Used only with `-t`. Specifies the position on the tape device where the archive should be created. If not specified, the current position of the tape device is used.

`-t` Creates an archive on or reads an archive from a tape device. The *archive* operand (see OPERANDS) is assumed to be the name of the tape device.

EXAMPLES

EXAMPLE 1 Creating a Flash Archive

The following command creates a flash archive named `pogoS9` and stores it in `/export/home/archives/s9fcs.flar`. The currently running system is the basis for the new archive.

```
# flar create -n pogoS9 /export/home/archives/s9fcs.flar
```

OPERANDS

The following operand is supported:

archive Path to tape device if the `-t` option was used. Otherwise, the complete path name of a flash archive. A file containing a flash archive has a standard file extension of `.flar`.

EXIT STATUS

The following exit values are returned for the `create`, `split`, and `combine` subcommands:

`0` Successful completion.

0 An error occurred.

The following exit values are returned for the `info` subcommand:

0 Successful completion.

1 Command failed. If the `-k` option is used and the requested keyword is not found, `flar` returns `2`.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWinst

SEE ALSO

`compress`(1), `flarcreate`(1MCM), `flash_archive`(4CM), `attributes`(5)

NAME

flarcreate – create a flash archive from a master system

SYNOPSIS

`flarcreate` `-n` *name* `[-R` *root*`] [-H] [-I] [-S] [-c] [-t [-p` *posn*`] [-b` *blocksize*`]] [-i` *date*`] [-u` *section*...`] [-U` *key=value*...`] [-m` *master*`] [-f [`*filelist* `| -] [-F]] [-a` *author*`] [-e` *descr* `| -E` *descr_file*`] [-T` *type*`] [-x` *exclude*...`] [-y` *include*...`] [-z` *filelist*...`] [-X` *filelist*...`]` *archive*

DESCRIPTION

The `flarcreate` command creates a flash archive from a master system. A master system is one that contains a reference configuration, which is a particular configuration of the Solaris operating environment, plus other optional software. A flash archive is an easily transportable version of the reference configuration.

In flash terminology, a system on which an archive is created is called a *master*. The system image stored in the archive is deployed to systems that are called *clones*.

An archive contains all the files that are in a system image.

Following the creation of a flash archive, you can use custom JumpStart to clone the archive on multiple systems.

You can run `flarcreate` in multiuser or single-user mode. You can also use the command when the master system is booted from the first Solaris software CD or from a Solaris net image.

Archive creation should be performed when the master system is in as stable a state as possible. Following archive creation, use the `flar`(1MCM) command to administer a flash archive.

See `flash_archive`(4CM) for a description of the flash archive.

The `flarcreate` command requires superuser privileges.

OPTIONS

The `flarcreate` command has one required option:

`-n` *name*
Specifies the name of the flash archive. *name* is supplied as the value of the `content_name` keyword. See `flash_archive`(4CM).

The `flarcreate` command has the following general options:

`-c`
Compresses the archive using `compress`(1).

`-f` *filelist*
Uses the contents of *filelist* as a list of files to include in the archive. The files are included in addition to the usual file list, unless `-F` is specified (see the `-F` option). If *filelist* is `-`, the list is taken from standard input.

`-F`
Includes only files in the list specified by `-f`. This option makes `-f` *filelist* an absolute list, rather than a list that is appended to the usual file list.

`-H`
Does not generate a hash identifier.

`-I`

Ignores integrity check. To prevent you from excluding important system files from an archive, `flarcreate` runs an integrity check. This check examines all files registered in a system package database and stops archive creation if any of them are excluded. Use this option to override this integrity check.

`-R` *root*

Creates the archive from the file system tree mounted at *root*. If you do not specify this option, `flarcreate` creates an archive from a file system mounted at `/`.

`-S`

Skips the disk space check. Without `-S`, `flarcreate` builds a compressed archive in memory before writing the archive to disk, to ensure you have sufficient disk space. Use `-S` to skip this step. The result of the use of `-S` is a significant decrease in the time it takes to create an archive.

`-U` *key=value...*

Includes one or more user-defined keywords and their values in the archive identification section. See `flash_archive`(4CM).

`-x` *exclude...*

Excludes the file or directory *exclude* from the archive. Note that the *exclude* file or directory is assumed to be relative to the alternate *root* specified using `-R`. If the parent directory of the file *exclude* is included with the `-y` option (see `-y` *include*), then only the specific file or directory specified by *exclude* is excluded. Conversely, if the parent directory of an included file is specified for exclusion, then only the file *include* is included. For example, if you specify:

```
-x /a -y /a/b
```

all of `/a` except for `/a/b` is excluded. If you specify:

```
-y /a -x /a/b
```

all of `/a` except for `/a/b` is included.

`-X` *filelist...*

Uses the contents of *filelist* as a list of files to exclude from the archive. If *filelist* is `–`, the list is taken from standard input.

`-y` *include...*

Includes the file or directory *include* in the archive. Note that the *exclude* file or directory is assumed to be relative to the alternate *root* specified using `-R`. See the description of the `-x` option for a description of the interaction of the `-x` and `-y` options.

`-z` *filelist*... Is a list of files prefixed with a plus (+) or minus (`-`). A plus indicates that a file should be included in the archive. The minus indicates exclusion. If *filelist* is `-`, the list is taken from standard input.

Use the following options with user-defined sections:

`-d` *dir* Retrieves the section file specified with `-u` from *dir*.

`-u` *section*... Includes the user-defined section located in the file *section* in the archive. *section* must be a blank-separated list of section names as described in `flash_archive`(4CM).

Use the following options with tape archives:

`-b` *blocksize* Is the block size to be used when creating the archive. If not specified, a default block size of 64K is used.

`-p` *posn* Used only with `-t`. Specifies the position on the tape device where the archive should be created. If not specified, the current position of the tape device is used.

`-t` Creates an archive on a tape device. The *archive* operand (see OPERANDS) is assumed to be the name of the tape device.

The following options are used for archive identification:

`-a` *author* Provides an author name for the archive identification section. If you do not specify `-a`, no author name is included in the identification section.

`-e` *descr* Is the description to be included in the archive as the value of the `content_description` archive identification key. This option is incompatible with `-E`.

`-E` *descr_file* Is the description to be used as the value of the archive identification `content_description` key as retrieved from the file *descr_file*. This option is incompatible with `-e`.

`-i` *date* By default, the value for the `creation_date` field in the identification section is generated automatically, based on the current system time and date. If you specify the `-i` option, *date* is used instead.

`-m` *master* By default, the value for the `creation_master` field in the identification section is the name of the system on which you run `flarcreate`, as reported by `uname -n`. If you specify `-m`, *master* is used instead.

`-T` *type* Is the content type included in the archive as the value of the `content_type` archive identification key. If you do not specify `-T`, the `content_type` keyword is not included.

OPERANDS

The following operand is supported:

archive Path to tape device if the `-t` option was used. Otherwise, the complete path name of a flash archive. A file containing a flash archive has a standard file extension of `.flar`.

EXIT STATUS

The following exit values are returned:

`0` Successful completion.

`0` An error occurred.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWinst

SEE ALSO

`compress`(1), `flar`(1MCM), `flash_archive`(4CM), `attributes`(5)

Change Manager File Formats

NAME

bart_manifest – system audit manifest file

DESCRIPTION

The `bart`(1MCM) command generates a manifest that describes the contents of a managed host. A manifest consists of a header and entries. Each entry represents a single file. Entries are sorted in ascending order by file name. Any nonstandard file names, such as those that contain embedded newline or tab characters, have the special characters quoted prior to being sorted. See Quoting Syntax.

Lines that begin with `!` supply metadata about the manifest. The manifest version line indicates the manifest specification version. The date line shows the date on which the manifest was created, in `date`(1) form.

Some lines are ignored by the manifest comparison tool. Ignored lines include blank lines, lines that consist only of white space, and comments that begin with `#`.

In addition to metadata lines, the header contains the format comment block. This comment block lists the attributes reported for each file type.

To see the format of an manifest file, see EXAMPLES.

Manifest File Entries

Each manifest file entry is a single line of one of the following forms, depending on the file type:

```
fname D size mode acl dirmtime uid gid  [xattr xcontents] *
fname P size mode acl mtime uid gid  [xattr xcontents] *
fname S size mode acl mtime uid gid  [xattr xcontents] *
fname F size mode acl mtime uid gid contents  [xattr xcontents] *
fname L size mode acl lnmtime uid gid dest  [xattr xcontents] *
fname B size mode acl mtime uid gid devnode  [xattr xcontents] *
fname C size mode acl mtime uid gid devnode  [xattr xcontents] *
```

Each entry begins with *fname*, which is the name of the file. To prevent parsing problems that are caused by special characters embedded in file names, file names are encoded as described in Quoting Syntax.

Subsequent fields represent the following file attributes.

type
Type of file. Possible values are as follows:

- `B` for a block device node
- `C` for a character device node
- `D` for a directory
- `F` for a file
- `L` for a symbolic link
- `P` for a pipe
- `S` for a socket

size
File size in bytes.

mode
Octal number that represents the permissions of the file.

acl
ACL attributes for the file. For a file with ACL attributes, this field contains the output from `acltotext()`.

uid
Numerical user ID of the owner of this entry.

gid
Numerical group ID of the owner of this entry.

dirmtime, *lnmtime*, *mtime*
Last modification time, in seconds since 00:00:00 UTC, January 1, 1970, for directories, links, and other files, respectively.

contents
Checksum value of the file. This attribute is only specified for regular files. If you turn off context checking or if checksums cannot be computed, the value of this field is `-`.

dest
Destination of a symbolic link.

devnode
Value of the device node. This attribute is for character device files and block device files only.

[*xattr xcontents*]*
Zero or more checksum values for files with extended attributes. The attributes are described in alphabetical order. If you specify the `-n` option or the `IGNORE contents` directive, the value of *xcontents* is `-`.

Quoting Syntax

The rules file supports a quoting syntax for representing nonstandard file names.

When generating an manifest for file names that embed tab, space, or newline characters, the special characters are encoded in their octal forms.

Input Character	Quoted Character
(space)	\(space)
(tab)	\(tab)
(newline)	\(newline)
?	\?
[	\[
*	*

EXAMPLES

EXAMPLE 1 Sample Manifest Output

The following is a sample system manifest file. The file entries are sorted by the encoded versions of the file names to correctly handle special characters.

```
! Version 1.0
! Mon Feb 11 10:55:30 2002
# Format:
```

EXAMPLE 1 Sample Manifest Output *(Continued)*

```
# fname D size mode acl dirmtime uid gid [xattr xcontents]*
# fname P size mode acl mtime uid gid [xattr xcontents]*
# fname S size mode acl mtime uid gid [xattr xcontents]*
# fname F size mode acl mtime uid gid contents [xattr xcontents]*
# fname L size mode acl lnmtime uid gid dest [xattr xcontents]*
# fname B size mode acl mtime uid gid devnode [xattr xcontents]*
# fname C size mode acl mtime uid gid devnode [xattr xcontents]*
/etc D 3584 40755 user::rwx,group::r-x,mask::r-x,other::r-x, 3c6803d7 0 3
/etc/.login F 524 100644 user::rw-,group::r--,mask::r--,other::r--, 3c165878
    0 3 27b53d5c3e844af3306f1f12b330b318
/etc/.pwd.lock F 0 100600 user::rw-,group::---,mask::---,other::---, 3c166121
    0 0 d41d8cd98f00b204e9800998ecf8427e
/etc/.syslog_door L 20 120777 user::rw-,group::r--,mask::rwx,other::r--,
    3c6803d5 0 0 /var/run/syslog_door
/etc/autopush L 16 120777 user::r-x,group::r-x,mask::r-x,other::r-x, 3c165863
    0 0 ../sbin/autopush
/etc/cron.d/FIFO P 0 10600 user::rw-,group::---,mask::---,other::---, 3c6803d5
    0 0
```

SEE ALSO

date(1), bart(1MCM), bart_rules(4CM), attributes(5)

NAME

bart_rules – bart rules file

DESCRIPTION

The rules file is a text file that is used by the `bart`(1MCM) command. The rules file determines which files to validate and which file attributes of those files to ignore.

Some lines are ignored by the manifest comparison tool. Ignored lines include blank lines, lines that consist only of white space, and comments that begin with `#`.

The rules file supports three directives: `CHECK`, `IGNORE`, and a subtree directive, which is an absolute path name and optional pattern matching modifiers. The rules file uses the directives to create logical blocks.

Syntax

The syntax for the rules file is as follows:

```
[IGNORE attribute...]*
[CHECK] [attribute...]*

subtree1 [pattern...]*
[IGNORE attribute...]*
[CHECK] [attribute...]*

subtree2 [pattern...]*
subtree3 [pattern...]*
subtree4 [pattern...]*
[IGNORE attribute...]*
[CHECK] [attribute...]*
...
```

Rule Blocks

Rule blocks are composed of statements that are created by using directives and arguments. There are three types of blocks.

Global block

The first block in the file. The block is considered “global” if it specifies `CHECK` and `IGNORE` statements, but no previous subtree statement. A global block pertains to all subsequent blocks.

Local block

A block that specifies `CHECK` and `IGNORE` statements as well as a subtree directive. The rules in this block pertain to files and directories found in the specified subtree.

Heir block

A block that contains a null `CHECK` statement, no arguments. This block inherits the global `CHECK` statements and `IGNORE` statements.

Note – The order in which `CHECK` and `IGNORE` statements appear in blocks is important. The `bart` command processes `CHECK` and `IGNORE` statements in the order in which they are read, with later statements overriding earlier statements.

Subtree specifications must appear one per line. Each specification must begin with an absolute path name. Optionally, each specification can be followed by pattern-matching arguments.

When a file being tracked belongs to more than one subtree directive, `bart` performs the following resolution steps:

- Applies the CHECK and IGNORE statements set in the global block. Note that all CHECK and IGNORE statements are processed in order.
- Finds the last subtree directive that matches the file.
- Processes the CHECK and IGNORE statements that belong to the last matching subtree directive. These statements are processed in the order in which they are read, overriding global settings.

Pattern Matching Statements

AND Statement

For a given subtree directive, all pattern matching statements are logically ANDed with the subtree. Patterns have the following syntax:

- Wildcards are permitted for both the subtree and pattern matching statements.
- The exclamation point (!) character represents logical NOT.
- A pattern that terminates with a slash is a subtree. The absence of a slash indicates that the pattern is not a directory. The subtree itself does not require an end slash.

For example, the following subtree example includes the contents of /home/nickiso/src except for object files, core files, and all of the SCCS subtrees. Note that directory names that terminate with .o and directories named core are *not* excluded because the patterns specified do not terminate with /.

```
/home/nickiso/src !*.o !core !SCCS/
CHECK    all
```

OR Statement

Group multiple subtree directives together. Such subtree directives are logically ORed together.

```
/home/nickiso/src !*.o !core
/home/nickiso/Mail
/home/nickiso/docs *.sdw
CHECK    all
IGNORE    mtime lnmtime dirmtime
```

The files included in the previous example are as follows:

- Everything under /home/nickiso/src except for *.o and core files
- Everything under /home/nickiso/Mail
- All files under /home/nickiso/docs that end in *.sdw

For these files, all attributes are checked except for modification times.

File Attributes

The bart command uses CHECK and IGNORE statements to define which attributes to track or ignore. Each attribute has an associated keyword.

The attribute keywords are as follows:

- acl
- all

- contents
- dest
- devnode
- dirmtime
- gid
- lnmtime
- mode
- mtime
- size
- type
- uid
- xattrs

The all keyword refers to all file attributes. See bart_manifest(4CM).

EXAMPLES

EXAMPLE 1 Sample Rules File

```
# Global rules, track everything except dirmtime.
CHECK    all
IGNORE    dirmtime

# The files in /data* are expected to change, so don’t bother
# tracking the attributes expected to change.
# Furthermore, by specifying ‘‘IGNORE contents,’’ you save
# time and resources.
/data*
IGNORE    contents mtime size

/home/nickiso f* bar/
IGNORE    acl

# For /usr, apply the global rules.
/usr
CHECK

# Note: Since /usr/tmp follows the /usr block, the /usr/tmp
# subtree is subjected to the ‘‘IGNORE all.’’
/usr/tmp
/home/nickiso *.o
/home/nickiso core
/home/nickiso/proto
IGNORE    all
```

The following files are cataloged based on the sample rules file:

- All attributes, except for dirmtime, mtime, size, and contents, are tracked for files under the /data* subtrees.
- Files under the /usr subtree, except for /usr/tmp, are cataloged by using the global rules.
- If the /home/nickiso/foo.c file exists, its attributes, except for acl and dirmtime, are cataloged.

EXAMPLE 1 Sample Rules File *(Continued)*

- All .o and core files under /home/nickiso, as well as the /home/nickiso/proto and /usr/tmp subtrees, are ignored.
- If the /home/nickiso/bar/foo.o file exists, it is ignored because it is subject to the last block.

SEE ALSO

bart(1MCM), bart_manifest(4CM), attributes(5)

NAME

cmsp – Sun Management Center Change Manager shared profile

DESCRIPTION

Shared profiles describe how one or more managed hosts are configured with a software stack. Much of the information described by these profiles is the same as described in an installation profile.

A shared profile file name must use the `.cmsp` suffix, for example, `web-server.cmsp`.

The shared profile is a set of properties and associated property values, one property per line. The property format is as follows:

property-name=property-value

Lines that contain only whitespace are ignored. Lines whose first non-whitespace character is # or ! are comments. The rest of the lines in the shared profile describe properties.

The property name consists of all the characters in the line starting with the first non-whitespace character and up to, but not including, the first equals sign (=) character.

The property value consists of the rest of the line after the equals sign.

If you want a backslash character to appear in the property value, escape the backslash with another backslash.

The following example shows that the value of the `base_config_target_arch` property is `sun4u`.

```
base_config_target_arch=sun4u
```

EXAMPLES

EXAMPLE 1 Default Shared Profile for Creating One Boot Environment

The following example shared profile uses the default values to create one boot environment.

```
#
# Example shared profile for a system with one boot environment.
#
# This example shared profile assumes a disk that is no smaller than
# 7 Gbytes in size.
#
# You must also specify the following properties with appropriate
# values:
#
# o base_config_flar_archive
#       Name of the Solaris Flash archive associated with this
#    shared profile
# o base_config_boot_image
#    Location of the Solaris boot image associated with the
#    specified Solaris Flash archive
# o base_config_sysidcfg_rootpw
```

EXAMPLE 1 Default Shared Profile for Creating One Boot Environment *(Continued)*

```
#       Encrypted root password entry, which can be taken from the
#    password entry in the /etc/shadow file
# o base_config_sysidcfg_timezone
#       Appropriate time zone value from /usr/share/lib/zoneinfo
#
base_config_target_arch=sun4u
base_config_sysidcfg_nameservice=none
base_config_sysidcfg_networkinterface=primary
base_config_sysidcfg_netmask=255.255.255.0
base_config_sysidcfg_ipv6=no
base_config_sysidcfg_defaultroute=none
base_config_sysidcfg_systemlocale=C
base_config_sysidcfg_terminal=vt100
base_config_sysidcfg_timeserver=localhost
base_config_sysidcfg_security_policy=none
base_config_be_0_root_device=rootdisk.s0
base_config_be_0_root_size=free
base_config_be_0_var_device=rootdisk.s3
base_config_be_0_var_size=1024
base_config_local_swap1_device=rootdisk.s1
base_config_local_swap1_size=2048
```

EXAMPLE 2 Default Shared Profile for Creating Two Boot Environments

The following example shared profile uses the default values to create two boot environments.

```
#
# Example shared profile for a system with two boot environments.
#
# This example shared profile assumes a disk that is no smaller than
# 12 Gbytes in size.
#
# You must also specify the following properties with appropriate
# values:
#
# o base_config_flar_archive
#       Name of the Solaris Flash archive associated with this
#    shared profile
# o base_config_boot_image
#    Location of the Solaris boot image associated with the
#    specified Solaris Flash archive
# o base_config_sysidcfg_rootpw
#       Encrypted root password entry, which can be taken from the
#    password entry in the /etc/shadow file
# o base_config_sysidcfg_timezone
#       Appropriate time zone value from /usr/share/lib/zoneinfo
#
base_config_target_arch=sun4u
base_config_sysidcfg_nameservice=none
base_config_sysidcfg_networkinterface=primary
base_config_sysidcfg_netmask=255.255.255.0
base_config_sysidcfg_ipv6=no
```

EXAMPLE 2 Default Shared Profile for Creating Two Boot Environments *(Continued)*

```
base_config_sysidcfg_defaultroute=none
base_config_sysidcfg_systemlocale=C
base_config_sysidcfg_terminal=vt100
base_config_sysidcfg_timeserver=localhost
base_config_sysidcfg_security_policy=none
base_config_be_0_root_device=rootdisk.s0
base_config_be_0_root_size=free
base_config_be_0_var_device=rootdisk.s3
base_config_be_0_var_size=1024
base_config_be_1_root_device=rootdisk.s4
base_config_be_1_root_size=4096
base_config_be_1_var_device=rootdisk.s5
base_config_be_1_var_size=1024
base_config_local_swap1_device=rootdisk.s1
base_config_local_swap1_size=2048
```

SEE ALSO

Sun Management Center Change Manager 1.0.1 Administration Guide

NAME

flash_archive – format of flash archive

SYNOPSIS

`flash_archive`

DESCRIPTION

A flash archive is an easily transportable version of a reference configuration of the Solaris operating environment, plus other optional software. Such an archive is used for the rapid installation of the Solaris software on large numbers of machines. The machine that contains a flash archive is referred to as a *master* system. A machine that receives a copy of a flash archive is called a *clone* system.

An archive is used for initial installation or whenever a complete, fresh installation is required. An archive contains all of the files from a master and overwrites the installed software on a clone completely.

You create a flash archive with the `flar`(1MCM) command or the `flarcreate`(1MCM) command. You view information about a given flash archive with `flar`. The `flar` command also enables you to split or combine the sections of a flash archive.

Flash archives are monolithic files that contain the following:

- Archive identification information
- Files that have been copied from a master system and that will be extracted onto a clone system

The standard extension for a file that contains a flash archive is `.flar`.

The flash archive is laid out in the following sections:

- Archive cookie
- Archive identification
- Predeployment
- Postdeployment
- Reboot
- Summary
- User-defined (optional)
- Archive files

The only assumptions that an application processing the archive can make about section number and placement is that there is an identification section located immediately after the archive cookie and that the last section in the archive is an archive files section.

These sections are described in the following subsections.

Archive Cookie Section

The very beginning of the archive contains a cookie, which serves to identify the file as a flash archive. It is also used by the deployment code for identification and validation purposes.

The case-sensitive, newline-terminated cookie that identifies version 1.*n* flash archives, is `FlAsH-aRcHiVe-1.`*n*, where *n* is an integer in the range 0 through 9.

The archive version is designed to allow for the future evolution of the flash archive specification while allowing applications that process flash archives to determine whether specific archives are of a format that can be handled correctly. The archive version is a number of the form *x.y*, where *x* is the major version number, and *y* is the minor version number.

When an application encounters a flash archive with an unknown major version number, it should issue an error message and exit.

Archive Identification Section

The archive identification section is plain text, delimited with newline characters. It is composed of a series of keyword and value pairs, with one pair allowed per line. Keywords and values are separated by a single equal sign. There are no limits to the length of individual lines. Binary data to be included as the value to a keyword is base64 encoded. The keywords themselves are case-insensitive. The case-sensitivity of the values is determined by the definition of the keyword, though most are case-insensitive.

The global order of the keywords within the identification section is undefined, except for the section boundary keywords. The identification section must begin with `section_begin=`*ident* and must end with `section_end=`*ident*.

In addition to the keywords defined for the flash archive and enumerated in the following table, users can define their own. These user-defined keywords are ignored by the flash mechanisms, but can be used by user-provided scripts or programs that process the identification section. User-defined keywords must begin with `X`, and contain characters other than linefeeds, equal signs, and null characters. For example, `X-department` is a valid user-defined keyword. `department`, which lacks the `X-` prefix, is not. Suggested naming conventions for user-defined keywords include the underscore-delimited descriptive method used for the pre-defined keywords, or a federated convention similar to that used to name Java packages.

Applications that process the identification section will process unrecognized non-user-defined keywords differently, depending on whether the archive version is known. If the application recognizes the archive specification version, it will reject any unrecognized non-user-defined keyword. If the application does not recognize the specification version, that is, if the minor version number is higher than the highest minor version it knows how to process, unrecognized non-user-defined keywords will be ignored. These ignored keywords are reported to the user by means of a non-fatal warning message.

Following are the keywords defined for this version of the Flash archive specification.

Keyword	Type of Value	Required
section_begin	Text	Yes
section_end	Text	Yes
archive_id	Text	No
files_archived_method	Text	No
files_compressed_method	Text	No
files_archived_size	Numeric	No
files_unarchived_size	Numeric	No
creation_date	Text	No
creation_master	Text	No
content_name	Text	Yes
content_type	Text	No
content_description	Text	No
content_author	Text	No
content_architectures	Text list	No
creation_node	Text	No
creation_hardware_class	Text	No
creation_platform	Text	No
creation_processor	Text	No
creation_release	Text	No
creation_os_name	Text	No
creation_os_version	Text	No

Future versions of the identification section might define additional keywords. The only guarantee regarding the new keywords is that they will not intrude upon the user-defined keyword namespace as shown previously.

Following is an example identification section:

```
section_begin=identification
files_archived_method=cpio
files_compressed_method=compress
files_archived_size=259323342
files_unarchived_size=591238111
creation_date=20000131221409
creation_master=pumbaa
content_name=Finance Print Server
```

```
content_type=server
content_description=Solaris 8 Print Server
content_author=Mighty Matt
content_architectures=sun4u
creation_node=pumbaa
creation_hardware_class=sun4u
creation_platform=SUNW,Sun-Fire
creation_processor=sparc
creation_release=5.9
creation_os_name=SunOS
creation_os_version=s81_49
x-department=Internal Finance
section_end=identification
```

Following are descriptions of the identification section keywords:

section_begin
section_end

These keywords are used to delimit sections in the archive and are not limited exclusively to the identification section. For example, the archive files section includes a section_begin keyword, though with a different value. User-defined archive sections will be delimited by section_begin and section_end keywords, with values appropriate to each section. The currently defined section names are given in the following table. User-defined names should follow the same convention as user-defined identification sections, with the additional restriction that they not contain forward slashes (/).

Section	Boundary
Identification	identification
Archive files	archive
Archive cookie	cookie

Note that while the archive cookie does not use section boundaries, and thus has no need for a section name within the archive itself, the flar(1MCM) command uses section names when splitting the archive, and thus requires a section name for the archive cookie. The name cookie is reserved for that purpose.

The following keywords, used in the archive identification section, describe the contents of the archive files section.

archive_id

This optional keyword *uniquely* describes the contents of the archive. It is computed as a unique hash value of the bytes representing the archive. Currently this value is represented as an ASCII hexadecimal 128-bit MD5 hash of the archive contents. This value is used by the installation software only to validate the contents of the archive during archive installation.

If the keyword is present, the hash value is recomputed during extraction based on the contents of the archive being extracted. If the recomputed value does not match the stored value in the identification section, the archive is deemed corrupt, and appropriate actions can be taken by the application.

If the keyword is not present, no integrity check is performed.

`files_archived_method`

This keyword describes the archive method used in the files section. If this keyword is not present, the files section is assumed to be in CPIO format with ASCII headers (the `-c` option to `cpio`). If the keyword is present, it can have the following value:

`cpio`	The archive format in the files section is CPIO with ASCII headers.

The compression method indicated by the `files_compressed_method` keyword (if present) is applied to the archive file created by the archive method.

The introduction of additional archive methods will require a change in the major archive specification version number, as applications aware only of `cpio` will be unable to extract archives that use other archive methods.

`files_compressed_method`

This keyword describes the compression algorithm (if any) used on the files section. If this keyword is not present, the files section is assumed to be uncompressed. If the keyword is present, it can have one of the following values:

`none`	The files section is not compressed.
`compress`	The files section is compressed using `compress`(1).

The compression method indicated by this keyword is applied to the archive file created by the archive method indicated by the value of the `files_archived_method` keyword (if any). `gzip` compression of the flash archive is not currently supported, as the `gzip` decompression program is not included in the standard miniroot.

Introduction of an additional compression algorithm would require a change in the major archive specification version number, as applications aware only of the above methods will be unable to extract archives that use other compression algorithms.

`files_archived_size`

The value associated with this keyword is the size of the archived files section, in bytes. This value is used by the deployment software only to give extraction progress information to the user. While the deployment software can easily determine the size of the archived files section prior to extraction, it cannot do so in the case of archive retrieval via a stream. To determine the compressed size when extracting from a stream, the extraction software would have to read the stream twice. This double read would result in an unacceptable performance penalty compared to the value of the information gathered.

If the keyword is present, the value is used only for the provision of status information. Because this keyword is only advisory, deployment software must be able to handle extraction of archives for which the actual file section size does not match the size given in `files_archive_size`.

If `files_archive_size` is not present and the archive is being read from a stream device that does not allow the prior determination of size information, such as a tape drive, completion status information will not be generated. If the keyword is not present and the archive is being read from a random-access device such as a mounted file system, or from a stream that provides size information, the compressed size will be generated dynamically and used for the provision of status information.

`files_unarchived_size`

This keyword defines the cumulative size in bytes of the extracted archive. The value is used for file system size verification. The following verification methods are possible using this approach:

No checking — If the `files_unarchived_size` keyword is absent, no checking for space will be performed.

Aggregate checking — If the `files_unarchived_size` keyword is present and the associated value is an integer, the integer will be compared against the aggregate free space created by the requested file system configuration.

The following keywords provide descriptive information about the archive as a whole. They are generally used to assist the user in archive selection and to aid in archive management. These keywords are all optional and are used by the deployment programs only to assist the user in distinguishing between individual archives.

`creation_date`

The value of the `creation_date` keyword is a textual timestamp representing the time of creation for the archive. The value of this keyword can be overridden at archive creation time through `flarcreate`(1MCM). The timestamp must be in ISO-8601 complete basic calendar format without the time designator (ISO-8601, §5.4.1(a)) as follows:

```
CCYYMMDDhhmmss
```

For example:

```
20000131221409
(January 31st, 2000 10:14:09pm)
```

The date and time included in the value should be in GMT.

`creation_master`

The value of the `creation_master` keyword is the name of the master machine used to create the archive. The value can be overridden at archive creation time.

content_name
The value of the content_name keyword should describe the archive's function and purpose. It is similar to the NAME parameter found in Solaris packages.

The value of the content_name keyword is used by the deployment utilities to identify the archive and can be presented to the user during the archive selection process, the extraction process, or both. The value must be no longer than 256 characters.

content_type
The value of this keyword specifies a category for the archive. This category is defined by the user and is used by deployment software for display purposes. This keyword is the flash analog of the Solaris packaging CATEGORY keyword.

content_description
The value of this keyword is used to provide the user with a description of what the archive contains and should build on the description provided in content_name. In this respect, content_description is analogous to the DESC keyword used in Solaris packages.

There is no length limit to the value of content_description. To facilitate display, the value can contain escaped newline characters. As in C, the escaped newline takes the form of \n. Due to the escaped newline, backlashes must be included as \\. The description is displayed in a non-proportional font, with at least 40 characters available per line. Lines too long for display are wrapped.

content_author
The value of this keyword is a user-defined string identifying the creator of the archive. Suggested values include the full name of the creator, the creator's email address, or both.

content_architectures
The value of this keyword is a comma-delimited list of the kernel architectures supported by the given archive. The value of this keyword is generated at archive creation time, and can be overridden by the user at that time. If this keyword is present in the archive, the extraction mechanism validates the kernel architecture of the clone system with the list of architectures supported by the archive. The extraction fails if the kernel architecture of the clone is not supported by the archive. If the keyword is not present, no architecture validation is performed.

The following keywords have default values that are filled in by uname(2) at the time the flash archive is created. If you create a flash archive in which the root directory is not /, the flash archive software inserts the string UNKNOWN for all of the creation_* keywords except creation_node, creation_release, and creation_os_name. For creation_node, the flash software uses the contents of the nodename(4) file. For creation_release and creation_os_name, the flash software attempts to use the contents of *<root_directory>*/var/sadm/system/admin/INST_RELEASE. If it is unsuccessful in reading this file, it assigns the value UNKNOWN.

Regardless of its source, you cannot override the value of a keyword that is filled in by uname.

`creation_node`
The return from `uname -n`.

`creation_hardware_class`
The return from `uname -m`.

`creation_platform`
The return from `uname -i`.

`creation_processor`
The return from `uname -p`.

`creation_release`
The return from `uname -r`.

`creation_os_name`
The return from `uname -s`.

`creation_os_version`
The return from `uname -v`.

Predeployment, Postdeployment, and Reboot Sections

Contain internal information that the flash software uses before and after deploying an operating environment image. These sections are for the exclusive use of the flash software.

Summary Section

Contains a summary of archive creation. This section records the activities of predeployment and postdeployment scripts.

User-Defined Sections

Following the identification section can be zero or more user-defined sections. These sections are not processed by the archive extraction code and can be used for any purpose.

User-defined sections must be line-oriented, terminated with newline (ASCII 0x0a) characters. There is no limit on the length of individual lines. If binary data is to be included in a user-defined section, it should be encoded using base64 or a similar algorithm.

Archive Files Section

The archive files section contains the files gathered from the master system. While the length of this section should be the same as the value of the `files_archived_size` keyword in the identification section, you should not assume that these two values are equal. This section begins with `section_begin=archive`, but it does not have an ending section boundary.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWinst

SEE ALSO

`compress`(1), `cpio`(1), `flar`(1MCM), `flarcreate`(1MCM), `md5`(3EXT), `attributes`(5)

NAME

ichange.cfg – Sun Management Center Change Manager configuration file

DESCRIPTION

You can change the behavior of the Change Manager application by modifying certain runtime parameters. These parameters are stored in the application configuration file, `ichange.cfg`. The configuration file is located in the `/var/opt/SUNWsymon/cfg` directory.

Note – When you make changes to the `ichange.cfg` file, you must restart the Sun Management Center services before the changes can take effect.

Restart the Sun Management Center services by running the following command as superuser:

```
# /opt/SUNWsymon/sbin/es-stop -S
# /opt/SUNWsymon/sbin/es-start -S
```

File Location Parameter

The `cmdataroot` parameter specifies the location of the Change Manager file hierarchy. `cmdataroot` points to the root of the Change Manager file hierarchy.

You might want to change the value of this parameter if you are moving the Change Manager repository to a different location.

The default value is the `/var/opt/ichange` directory.

Sun Management Center Agent Parameter

This is the Sun Management Center agent parameter:

`agentport`

Sun Management Center agent port to be used. Any update or reinstallation operations in which host parameters do not explicitly specify a value for the agent port will use this one.

The default value is `161`.

Job Execution Parameters

The following parameters describe job execution characteristics:

`boottimeout`

Interval to wait for a reboot, update, or reinstallation to complete. This is equivalent to the time it takes for the following events to occur:

- Complete the entire software installation, including any finish scripts
- The subsequent reboot to return
- Any boot-time startup procedures to run
- The Sun Management Center agent to reestablish communications with the management server

If the host does not reboot and reestablish agent communications with the Sun Management Center server within the specified time period, the associated management operation will fail with a timeout error.

You might need to change this value if any of the following are true:

- A managed host takes an unusually long time to boot.
- Your Sun Management Center topology requires a long time to establish server context for a newly configured agent.
- A particular software stack takes a long time to install.
- Finish and startup scripts take a long time to complete.

The default value is 1800000 milliseconds (30 minutes).

`downtimeout` Amount of time to wait for a managed host to shut down after a management operation has requested a reboot. This is effectively the time it takes for a system to complete an `init 6` sequence.

If the host does not shut itself down within the specified time, the associated management operation will fail with a timeout error. Thus, you might need to adjust this value if a host or software stack takes an unusually long time to complete its shutdown sequence.

The default value is 300000 milliseconds (5 minutes).

Other Parameters

`debug` Control the printing of debug messages to the Sun Management Center server console.

The default value is `false`. To turn on the debugging messages, change the value to `true`.

SEE ALSO

Sun Management Center Change Manager 1.0.1 Administration Guide