Fujitsu Siemens Computers Add-on Reference Manual

User Commands(1)

NAME

`acf -` array configuration

SYNOPSIS

`acf –i –z` *hw-addr*

`acf –c –z` *hw-addr –canned-config-key-letter*

`acf –l –z` *hw-addr* [`–f` *file*]

`acf –u –z` *hw-addr logical_unit_configuration_options*

`acf –e –z` *hw-addr error_recovery_configuration_options*

`acf –d –z` *hw-addr array_drive_configuration_options*

`acf –f –z` *hw-addr*

`acf –R –z` *hw-addr*

DESCRIPTION

`acf` is used to perform initial configuration of the disk array when installing a disk array on a system. It is also used to reconfigure the disk array whenever reconfiguration is required (e.g., addition of more disk drives, redefinition of LUNs, RAID levels, etc.). The first command option is taken as a function key letter, identifying the desired function:

`–i` Display device inquiry information

`–c` Execute a "canned" configuration sequence

`–l` Download array controller firmware

`–u` Set or display array logical unit parameters

`–e` Set or display array error recovery parameters

`–d` Set or display array drive parameters

`–f` Format a logical unit of a disk array

`–R` Establish the configuration of the array controller or the specified logical drive again.

Option `–z`:

`–z` The argument *hw-addr* is a string of characters giving the hardware address of the "object" to which the command is directed – this can be a controller, a logical unit, or a physical drive. *hw-addr* has exactly one of the following forms:

`iosb/sraid`*ppu*

`iosb/sraid`*ppu*`l`*l*

`iosb/sraid`*ppu*`:`*bp*

The first form specifies a controller, the second designates a logical unit, and the third one refers to a physical drive.

The naming conventions are as follows:

b RM400: 0 (on-board iobus) or 1 (peripheral bus)

RM600: always 0

pp RM400: the SCSI Host Adapter port number

RM600: the logical controller number

u the device's unit number

l the logical unit name; this may range from 0 to 7

: separator between array controller name and backend drive parts of *hw-addr*

After the separator, the naming conventions are as follows:

b specifies the SCSI bus of the array controller

p specifies the target ID of the array controller

Some examples of valid hardware address notation:

`ios0/sraid041`
system bus 0, host controller 04, unit number 1

`ios0/sraid041l0`
system bus 0, host controller 04, unit number 1, LUN 0

`ios0/sraid041:15`
system bus 0, host controller 04, unit number 1, physical drive on SCSI bus 1 and target id 5 of the array controller

OPTIONS

Options for canned configuration invocation:

–canned-config-key-letter
canned-config-key-letter is a letter that identifies an entry in a file of "canned" array controller configurations (`/etc/default/acf`). `acf` reads the file of canned configurations, locates the entry identified by *canned-config-key-letter* and invokes an associated sequence of configuration commands, passing *hw-addr* as an input to the sequence. (The **FILES** section below gives details on the organization of the `/etc/default/acf` file).

Options for array controller download:

`–f` *file*
Name of the file containing the disk array controller firmware, or else a list of files, each of which contains firmware to be downloaded. The object files must be in Motorola S-record format. If this option is not present, the controller firmware file is read from standard input.

Options for logical unit configuration:

(For all options listed below, a "d" instead of a value following the option letter will cause the current setting of the option to be displayed to standard out in the format "*name_of_parameter* = *value_of_parameter*".)

`–B d` Number of blocks in the LUN indicated by *hw-addr* (display only, unchangeable).

`–b` *n* Set the logical block size for the LUN indicated by *hw-addr* to *n*.

`–U` {`a` | `x` | `r` | `d`}
If "a" is present, the logical unit indicated by *hw-addr* is added; if "x" is present, the logical unit is deleted; if "r" is present, the status of the logical unit is revived; if "d" is present, the status of the logical unit is displayed.

`–d` *drive-list*
The drives in the *drive-list* are associated with the logical unit indicated by *hw-addr*. The drive list consists of drive identifiers (following the notation of the `–z` option), separated by commas. A drive identifier uses only the SCSI bus and target ID components of the hardware address to identify a drive in the array. (An example of a drive list representing the first rank of drives is: 10,20,30,40,50.)

Two groups of drive (disk ranks) must be specified for RAID-Level 1, where "ff" is used to separate the groups, e.g. `–d11,21,31,ff,12,22,32`. RAID-Level 1 is set implicitly with this format!

`–r` *n* Set the RAID level for the LUN indicated by *hw-addr* to *n*.

RAID-Level 1 is already set with the `–d` option (see above), if the corresponding drive list format is the

default!

`-S` *n* Set the segment size to *n* for the LUN indicated by *hw-addr*.

`-Z` *n* Set the segment zero size to *n* for the LUN indicated by *hw-addr*.

`-s d` Sector size for the LUN indicated by *hw-addr* (display only, unchangeable).

`-A` {`on`|`off`}
This option controls disabling of automatic drive-replacement-detection for the LUN indicated by *hw-addr*. `on` will disable automatic drive-replacement-detection, and `off` will enable it. When enabled (i.e., set to `off`), the controller will automatically detect a drive replacement and automatically begin reconstructing the drive.

`-F` *freq*
This option is the reconstruction frequency, or the number of tenths of a second to delay between instances of reconstruction activity.

`-a` *amount*
This option is the reconstruction amount, or the number of blocks to reconstruct once reconstruction activity is initiated.

`-t` {*numeric-mode-page* | *symbolic-mode-page*}
This option controls the type of mode pages that the array controller will sense when issuing a `MODE SENSE` command to the array drives. If the value following the `-t` option is a number, it is interpreted as a hexadecimal value; otherwise it is a symbolic page type. Choices for symbolic page types are:

`default`
sense default pages

`current`
sense current pages

`saved`
sense saved pages

`-M` {*numeric-status* | *symbolic-status*}
This option switches the specified controller to an active role and the alternative controller to a passive role or gives both controllers an active status.

If a number is entered after `-M`, it is interpreted as a hexadecimal status value.

Alternatively, a symbolic status can be specified:

`active`
Switches the specified controllers to an active status.

`dual` Switches both controllers to an active status.

If the "dual" status is activated, you have to specify a list of the logical drives, which are to be assigned to the specified controller. All other logical drives are assigned to the alternative controller. If the controllers are already in "dual" status, the specified logical drives are also assigned to this controller the next time this option is invoked.

The drive list can have the following format, for example:

`-Mdual:1,3,6` or `-M2:1,3,6`

`-m` {*numeric-status* | *symbolic-status*}
This option controls the redundant controller operational mode. If the value following the `-m` option is a number, it is interpreted as the hexadecimal value for the status; otherwise it is a symbolic status. Choices for the symbolic status and associated meanings are:

`release`
release failed controller from reset condition

`fail` fail alternate controller

Note: the order of the above options can be important in some cases as all changes to the configuration are done exactly in the order in which the options are given.

`-h` {`d` | `a##` | `x##`}
This option controls the configuration of hot spare drives:

`-hd` Displays the list of configured hot spare drives.

`-ha##`
Defines this drive (`##`, same format as with `-d`) as the hot spare drive.

`-hx##`
Marks the original hot spare drive as "free".

`-w` {`d` | `a` | `x`}
This option controls the setting of the "write cache" mode; it is valid for the specified logical drive.

`-wd` Displays the current setting (switched on/off).

`-wa` Switches on "write cache" mode.

`-wx` Switches off "write cache" mode.

`-i` {`d` | `a` | `x`}
This option controls the setting of the cache mirroring mode; it is valid for the specified logical drive.

`-id` Displays the current setting (switched on/off).

`-ia` Switches on "cache mirroring" mode. Note: This mode is only effective if the write cache is also switched on.

`-ix` Switches off "cache mirroring" mode.

Options for array error recovery configuration:

(For all options listed below, a d instead of a value following the option letter will cause the current setting of the option to be displayed to standard out in the format "*name_of_parameter* = *value_of_parameter*".)

`-r` *n* Set the read retry count to *n* for the controller indicated by *hw-addr*.

`-w` *n* Set the write retry count to *n* for the controller indicated by *hw-addr*.

`-W` {`on`|`off`}
Enable (on) or disable (off) automatic write reallocation for the controller indicated by *hw-addr* (display only, unchangeable).

`-R` {`on`|`off`}
Enable (on) or disable (off) automatic read reallocation for the controller indicated by *hw-addr* (display only, unchangeable).

`-P` {`on`|`off`}
Enable (on) or disable (off) reporting of recovered errors for the controller indicated by *hw-addr*.

Options for drive configuration:

(For all options listed below, a d instead of a value following the option letter will cause the current setting of the option to be displayed to standard out in the format "*name_of_parameter* = *value_of_parameter*".)

`-c` {*numeric-status* | *symbolic-status*}
Set the drive status. If the value following the `-c` option is a number, it is interpreted as the hexadecimal value for the status; otherwise it is a symbolic status. Choices for the symbolic status and associated meanings are:

`add` add drive

`delete`
mark drive as nonexistent

`fail` mark drive as failed

`replace`
mark drive as replaced and begin reconstruction

`replace_nf`

mark drive as replaced and begin reconstruction (without forcing a drive format)

Note: if `replace` is used, the drive is always formatted; if `replace_nf` (no format) is used, the drive is formatted only if the controller determines that a format is necessary.

FILES

The file `/etc/default/acf` is an ASCII file containing a series of identically-formed entries. Each entry associates three items of information: (1) a unique key letter for the entry (used to select a particular entry on the `acf` command line), (2) a text string describing the particular disk array configuration this entry corresponds to, and (3) the full pathname for a command that, when executed, will carry out the steps needed to set up the configuration. This command will typically be a (Bourne) shell script and must follow the convention of being able to accept exactly one argument, the hardware address in a form that matches the *hw-addr* argument to `acf`. A colon is used to delimit fields within an entry, as in the following example:

```
a:RAID level 0, 2 drives:/etc/setupR0-2
b:RAID level 5, 3 drives:/etc/setupR5-3
c:RAID level 0, 5 drives:/etc/setupR0-5
d:RAID level 5, 5 drives:/etc/setupR5-5
```

etc.

Each entry is contained on one line and may not span multiple lines.

EXIT STATUS

`0` = Error-free termination

`1` = Command line syntax error

`2` = Argument not in acceptable range

`3` = Error opening a file or device

`4` = SCSI-bus or SCSI-device-related error condition

`5` = Non-specific error

`6` = Insufficient memory

NOTES

In some cases, `acf` will require exclusive ownership of the hardware modules affected by the operations being performed, meaning, for example, that a logical unit of the array could not be accessed as a file system at the same time that its configuration was being fundamentally altered. `acf` does try to be as liberal as possible in terms of permitting those configuration actions that can be performed safely without exclusive ownership. If the program cannot acquire exclusive ownership when needed, it displays a notice to that effect and terminates.

`acf -l` will not work correctly if it encounters a Motorola S-record file having multiple end records.

Invoking `acf` with an unrecognized function keyword, as in

```
acf ?
```

will cause usage information to be displayed.

A current limitation of `acf` is that it is unable to configure a RAID 3 logical unit.

The ability to submit a list of files to be downloaded is not a supported feature.

With release 3.0 of the array controller software, the `add` argument to the `-c` option letter under drive configuration can be used to restore a drive in a failed or warning state back to the optimal state without reconstructing the drive. This function is intended for careful usage in multiple-drive-down situations and can result in loss of data and inconsistent array parity if the data on the logical unit has changed since the drive in question was placed in the failed or warning state.

After adding a LUN, a `dksetup -R` must be executed (unless the system is rebooted).

Switching the passive controller into the active role may take some time.

SEE ALSO

apc(1M), apr(1M), arc(1M), can(1M), lad(1M).

Fujitsu Siemens Computers Add-on Reference Manual

apc(1M)

NAME

`apc` - array parity check

SYNOPSIS

`apc −z` *hw-addr* [`−f` *file*]

DESCRIPTION

`apc` provides a means for identifying parity inconsistencies in the array that may have come about due to unrecovered power failures, system crashes, or controller failures.

The output of `apc` is a list of SCSI logical block addresses which were found to have parity inconsistencies.

The argument *hw-addr* is a string of characters giving the hardware address of the LUN to be checked. *hw-addr* has the following form:

`ios`*b*`/sraid`*pp*`u`*l*

The naming conventions are as follows:

b RM400: 0 (on-board iobus) or 1 (peripheral bus)
RM600: always 0

pp RM400: the SCSI Host Adaptor port number
RM600: the logical controller number

u the device's unit number

l the logical unit name; this may range from 0 to 7

Some examples of valid hardware address notation:

`ios0/sraid041l0`
system bus 0, host controller 04, unit number 1, LUN 0

`ios0/sraid043l0`
system bus 0, host controller 04, unit number 3, LUN 0

OPTIONS

`−f` *file*
Name of the output file to which the list of blocks which were found to have inconsistent parity is written. If this option is omitted, the list is written to standard output.

EXIT STATUS

`0 =` Error-free termination
`1 =` Command line syntax error
`2 =` Argument not in acceptable range
`3 =` Error opening a file or device
`4 =` SCSI-bus or SCSI-device-related error condition
`5 =` Non-specific error
`6 =` Insufficient memory
`8 =` Inappropriate device or system state

NOTES

Invoking `apc` with invalid arguments will cause usage information to be displayed.

SEE ALSO

acf(1M), apr(1M), arc(1M), can(1M), lad(1M).

Fujitsu Siemens Computers Add-on Reference Manual

apr(1M)

NAME

`apr` - array parity repair

SYNOPSIS

`apr –z` *hw-addr* [`–f` *file*]

DESCRIPTION

`apr` provides a means for repairing parity inconsistencies in the array that may have come about due to unrecovered power failures, system crashes, or controller failures.

The input to `apr` is a list of SCSI logical block addresses which were found to have parity inconsistencies.

The argument *hw-addr* is a string of characters giving the hardware address of the LUN to be repaired. *hw-addr* has the following form:

`ios`*b*`/sraid`*pp*`u`*l*

The naming conventions are as follows:

b RM400: 0 (on-board iobus) or 1 (peripheral bus)
RM600: always 0

pp RM400: the SCSI Host Adaptor port number
RM600: the logical controller number

u the device's unit number

l the logical unit name; this may range from 0 to 7

Some examples of valid hardware address notation:

`ios0/sraid041l0`
system bus 0, host controller 04, unit number 1, LUN 0

`ios0/sraid043l0`
system bus 0, host controller 04, unit number 3, LUN 0

OPTIONS

`–f` *file*
Name of the input file from which the list of blocks to be repaired is read. If this option is omitted, the list is read from standard input.

EXIT STATUS

`0 =` Error-free termination
`1 =` Command line syntax error
`2 =` Argument not in acceptable range
`3 =` Error opening a file or device
`4 =` SCSI-bus or SCSI-device-related error condition
`5 =` Non-specific error
`6 =` Insufficient memory
`8 =` Inappropriate device or system state

NOTES

Invoking `apr` with invalid arguments will cause usage information to be displayed.

With release 3.0 of the array controller software, it is not necessary for `apr` to acquire exclusive access to the logical unit to be repaired, meaning parity repairs can be done without requiring that filesystems be

unmounted. If the array controller is running an earlier version of software, it will still be necessary to unmount filesystems before attempting to repair parity.

When fixing parity by the method of reading and re-writing the data, `apr` will go ahead and re-write the data even if an error was returned on the read.

SEE ALSO

`acf`(1M), `apc`(1M), `arc`(1M), `can`(1M), `lad`(1M).

Fujitsu Siemens Computers Add-on Reference Manual

arc(1M)

NAME

`arc -` array reconstruction

SYNOPSIS

`arc -z` *hw-addr* [`-P`] [`-s`] [`-F` {`d`|*freq*}] [`-a` {`d`|*amount*}]

DESCRIPTION

`arc` provides a means for controlling the reconstruction of data on disk drives which have been replaced or otherwise repaired after having been taken off-line.

The first argument, *hw-addr*, is a string of characters giving the hardware address of the "object" to which the command is directed – this can be a logical unit or a physical drive. *hw-addr* has exactly one of the following forms:

`ios`*b*`/sraid`*ppu*`l`*l*

`ios`*b*`/sraid`*ppu*`:`*bp*

The first form specifies a logical unit, the second one refers to a physical drive.

The naming conventions are as follows:

b RM400: 0 (on-board iobus) or 1 (peripheral bus)
RM600: always 0

pp RM400: the SCSI Host Adaptor port number
RM600: the logical controller number

u the device's unit number

l the logical unit name; this may range from 0 to 7

`:` separator between array controller name and backend drive parts of *hw-addr*

After the separator, the naming conventions are as follows:

b specifies the SCSI bus of the array controller

p specifies the target ID of the array controller

Some examples of valid hardware address notation:

`ios0/sraid041l0`
system bus 0, host controller 04, unit number 1, LUN 0

`ios0/sraid041:15`
system bus 0, host controller 04, unit number 1, physical drive on SCSI bus 1 and target id 5 of the array controller

OPTIONS

`-P` This option displays the progress of the reconstruction as a percentage. (For this option, the hardware address must refer to a logical unit, as in `ios`*b*`/sraid`*ppu*`l`*l*.)

`-s` This option notifies the array controller of a drive replacement and initiates the reconstruction. (For this option, the hardware address must refer to a drive, as in `ios`*b*`/sraid`*ppu*`:`*bp*.)

`-F` {`d`|*freq*}
This argument is the reconstruction frequency, or the number of tenths of a second to delay between instances of reconstruction activity. If "d" is specified, the current reconstruction frequency is displayed; if *freq* is specified, the reconstruction frequency is set to *freq*. (For this option, the hardware address must refer to a logical unit, as in `ios`*b*`/sraid`*ppu*`l`*l*.)

`-a` {`d`|*amount*}
This argument is the reconstruction amount, or the number of blocks to reconstruct once reconstruction

activity is initiated. If "d" is specified, the current reconstruction amount is displayed; if *amount* is specified, the reconstruction amount is set to *amount*. (For this option, the hardware address must refer to a logical unit, as in `iosb/sraidppull`.)

EXIT STATUS

`0` = Error-free termination
`1` = Command line syntax error
`2` = Argument not in acceptable range
`3` = Error opening a file or device
`4` = SCSI-bus or SCSI-device-related error condition
`5` = Non-specific error
`6` = Insufficient memory

NOTES

Invoking `arc` with invalid arguments will cause usage information to be displayed.

SEE ALSO

`acf`(1M), `apc`(1M), `apr`(1M), `can`(1M), `lad`(1M).

Fujitsu Siemens Computers Add-on Reference Manual

can(1M)

NAME

`can` - convert array name

SYNOPSIS

`can` *device*

DESCRIPTION

`can` accepts a UNIX device path name as its single argument and converts it to the "normal form" used by the disk array utilities (e.g., see the description of the `-z` argument for the `acf` utility.) The converted name is written to standard output.

EXIT STATUS

`0` = Error-free termination

`1` = Command line syntax error

`3` = Error opening a file or device

`5` = Non-specific error

NOTES

Invoking `can` with no arguments will cause usage information to be displayed.

SEE ALSO

`acf`(1M), `apc`(1M), `apr`(1M), `arc`(1M), `lad`(1M).

Fujitsu Siemens Computers Add-on Reference Manual

cc(1-ucb)
BSD Compatibility Package

NAME

`cc` - C compiler

SYNOPSIS

`/usr/ucb/cc` [*options*]

DESCRIPTION

`/usr/ucb/cc` is the C compiler for the BSD Compatibility Package. `/usr/ucb/cc` is identical to `/usr/bin/cc` [see `cc`(1)] except that BSD header files and BSD libraries are linked *before* System V libraries.

`/usr/ucb/cc` accepts the same options as `/usr/bin/cc`, with the following exceptions:

`-I"`*dir*`"`
: Search *dir* for included files whose names do not begin with a "`/`", prior to the usual directories. The directories for multiple `-I` options are searched in the order specified. The preprocessor first searches for `#include` files in the directory containing *sourcefile*, and then in directories named with `-I` options (if any), then `/usr/ucbinclude`, and finally, in `/usr/include`.

`-L"`*dir*`"`
: Add *dir* to the list of directories searched for libraries by `/usr/ccs/bin/cc`. This option is passed to `/usr/ccs/bin/ld`. Directories specified with this option are searched before `/usr/ucblib`, `/usr/ccs/lib` and `/usr/lib`.

`-Y LU,`*dir*
: Change the default directory used for finding libraries.

NOTES

The `-Y LU,`*dir* option may have unexpected results, and should not be used. This option is not in the UNIX System V base.

FILES

`/usr/ucblib`
`/usr/lib`
`/usr/ucblib/libucb.a`
`/usr/ccs/lib`
`/usr/ucb`
`/usr/ccs/bin`

SEE ALSO

`ar`(1), `as`(1), `cc`(1), `ld`(1), `lorder`(1), `strip`(1), `tsort`(1), `a.out`(4).

Fujitsu Siemens Computers Add-on Reference Manual

clmstat(1)

NAME

`clmstat -` report Cluster Manager statistics

SYNOPSIS

`clmstat` –a [–g *group*] [–v] [–D] [*interval count*]

`clmstat` –n *nodename* [–g *group*] [–v] [–D] [*interval count*]

`clmstat` –g *groupname* [–v] [–D] [*interval count*]

`clmstat` [–v] [–D] [*interval count*]

DESCRIPTION

`clmstat` displays CLM (Cluster Manager) statistics, including groups in the cluster, members for each group, and status for the members. `clmstat` always defaults to the current node if the node name is not specified. To retrieve statistics for a remote node from other nodes within the cluster, a node name must be specified. The `–n` option is used to specify the node name.

Group statistic collected by `clmstat` without options or with `–n` option is the groups created by this node, and the members statistic collected for these groups is members created by this node also. Therefore, these statistics collected from local node are not necessary the total groups and members in the whole cluster.

For remote node's statistic, user can use `–n` to give node name, or use `–a` option to show all the nodes in the cluster.

`–g` option will show all the members of the given group name, no matter on which node they are attached.

CLM statistics can be monitored by specifying an interval in seconds. A running display of statistics accumulated over each interval is produced. For example, `clmstat` 5 will print the accumulated statistics for the local node every five seconds. If a count is given, the statistics are repeated count times.

OPTIONS

`–a` Print CLM statistics for all the nodes in the cluster.

`–n` Print CLM statistics for node. Node name is required.

`–g` Print CLM statistics for group. Group name is required.

`–v` Verbose statistics output.

interval
Time interval between each executing `clmstat`. Must be used together with count.

count
Times for executing `clmstat`, must be used together with interval.

Fujitsu Siemens Computers Add-on Reference Manual

dlmstat(1)

NAME

`dlmstat -` report Distributed Lock Manager statistics

SYNOPSIS

`dlmstat [ -a ]`

`dlmstat [ -l` *line* `] [ -n` *nodename* `] [` *interval* `] [` *count* `]`

DESCRIPTION

`dlmstat` displays Distributed Lock Manager memory allocation statistics. The statistics display the number of configured and in-use structures for the data types `process`, `resource`, `lock`, `message buffer`, `info`, `transaction`, and `group`. These data structures are reported on a per-node basis.

`dlmstat` also calculates the size of memory allocated to hold these data structures.

The optional trailing *interval* causes `dlmstat` to report the statistics every interval second for *count* times. If *count* is omitted, the output is repeated continuously.

OPTIONS

`-a` Print DLM statistics for all the nodes in the cluster.

`-n` *nodename*
: Print DLM statistics for *nodename*.

`-l` *line*
: Generate output at *interval*, with *line* iterations between printed banners. If *line* is 0, the banner is displayed only once when the program starts.

SEE ALSO

`dlmconfig`(8) .

Fujitsu Siemens Computers Add-on Reference Manual

icfstat(1)

NAME

`icfstat -` report Internode Communication Facility statistics

SYNOPSIS

`icfstat [ -a ]`

`icfstat [ -n` *nodename* `] [` *interval* `[`*count* `]]`

`icfstat [ -S ]`

`icfstat [ -s ]`

`icfstat [ -V ]`

`icfstat [ -z ]`

DESCRIPTION

`icfstat` displays ICF (Internode Communication Facility) statistics collected per node since the last time they were cleared, including round trip timing histograms, packet size histograms, verbose statistics, and per-service statistics.

`icfstat` always defaults to the current node if the node name is not specified. To retrieve statistics for a remote node from other nodes within the cluster, a node name must be specified. The `-n` option is used to specify the node name. Currently not all the statistics is available with the `-n` option.

ICF statistics can be monitored by specifying an *interval* in seconds. A running display of statistics accumulated over each interval is produced. For example, `icfstat 5` will print the accumulated statistics for the local node every five seconds. If a count is given, the statistics are repeated *count* times. This option is limited to certain statistics.

OPTIONS

- `-a` Print ICF statistics for all the nodes in the cluster. For each of the directions transmit and receive three fields are printed: `Total` is the total number of ICF packets transferred, `Cpkts` is the number of control packets transferred and `DMAs` is the number of packets transferred in DMA mode.
- `-n` ICF statistics for node name *nodename* every *interval* seconds for *count* iterations. The first line and every 21st line gives the total values for the statistics. The intermediate lines give delta values compared to the previous one.
- `-S` Histogram of ICF packet sizes (transmit and receive).
- `-s` Per-service ICF statistics for local node.
- `-V` Verbose statistics between local node and rest of the nodes.
- `-z` Clear all statistics on local node.

SEE ALSO

`icfmap`(1M) .

Fujitsu Siemens Computers Add-on Reference Manual

intro(1)

NAME

`intro` - introduction to the intro manpages

DESCRIPTION

With SINIX V5.42, the names of the various `intro` manpages were changed in order to make them more logical. Up to SINIX V5.41, the call

```
man intro
```

displayed 10 different `intro` manpages from three different manuals to the user. There was an `intro` manpage from the former "User's Reference Manual" (URM), that is now called "Commands. User's Reference Manual". There were also `intro` manpages for individual chapters of the "Programmer's Reference Manual" (PRM), and of the "System Administrator's Reference Manual" (SARM).

In order to dispense with the variety of names and to allow users to call particular descriptions directly, each `intro` manpage was assigned a name extension indicating the manual and chapter to which it belongs. An attempt was made to use this convention for the older `intro` manpages, however inconsistently. The following is an overview of the previous and the current status.

	Old name	New name	
Manual	Up to V5.41	V5.42 or later	Chapters affected
URM	`intro`	`intro_urm`	Ch. 1: User Commands
PRM	`intro_ccs`	`intro_prm1`	Ch. 1: Programmer Commands
	`intro`	`intro_prm2`	Ch. 2: System Calls
	`intro`	`intro_prm3`	Ch. 3: Functions of Library
	`intro`	`intro_prm3m`	Ch. 3M: Math Library
	`intro_prm`	`intro_prm4`	Ch. 4: File Formats
	`intro`	`intro_prm5`	Ch. 5: Miscellaneous Functions
SARM	`intro`	`intro_sarm1`	Ch. 1M: System Administrator Commands
	`intro`	`intro_sarm4`	Ch. 4: File Formats
	`intro`	`intro_sarm5`	Ch. 5: Miscellaneous Functions
	`intro`	`intro_sarm7`	Ch. 7: Special Files
	`intro`	`intro_sarm8`	Ch. 8: System Maintenance
NETS *	–	`intro_net1`	Ch. 1: User Commands

* NETS = "Networking Reference Manual" [new with SINIX V5.43].

The individual `intro` manpages differ in quality and content. Some are very extensive, such as the `intro` manpage for chapters 2 and 3 of the PRM. This is a new manpage, in which the former descriptions of system calls, library functions, and the math library have been summarized in a single manpage. In other words: regardless of whether the user calls

```
man intro_prm2
```

or

```
man intro_prm3
```

or

`man intro_prm3m`

the same `intro` manpage is displayed, and the user must then search through it to find the relevant passages (note: for a detailed description of the `more(1)` pager, which simplifies searching, see the manpage of the same name or use the online call `man more`). The online version of each `intro` manpage is identical to the printed version in the relevant manual, which is convenient for looking up longer descriptions in the manual.

Since the old `intro` manpages were replaced by those with name extensions, the calls

`man intro`

and

`man 1 intro`

will display only this general manpage to the user. All other `intro` manpages must be called using their new, extended names, e.g.

`man intro_sarm1`

etc.

SEE ALSO

`more(1)` .

Fujitsu Siemens Computers Add-on Reference Manual

lad(1M)

NAME

`lad` - list array devices

SYNOPSIS

`lad` [`–n` *string*]

DESCRIPTION

`lad` will find all of the disk array logical units attached to the system and write their associated UNIX device name (e.g., `/dev/ios`*b*`/sraid`*ppul*) to standard output, one device per line.

OPTIONS

`–n` *string*

Include in the output list the first non-existent logical unit of each existing controller. Each non-existent logical unit in the output display is "flagged" by appending *string* to the line, separating it from the device name with a space.

EXIT STATUS

`0` = Error-free termination

`1` = Command line syntax error

NOTES

Invoking `lad` with invalid arguments will cause usage information to be displayed.

SEE ALSO

`acf`(1M), `apc`(1M), `apr`(1M), `arc`(1M), `can`(1M).

man-news(1)

NAME

`man-news -` new or updated manual pages with Release 5.45B

DESCRIPTION

Three changes have been made to manual pages between Release Reliant UNIX 5.45A and Release Reliant UNIX 5.45B:

- New manual pages have been included
- Some manual pages have been removed
- Some manual pages have been modified or corrected

The following alphabetically ordered table indicates the particular category ("New", "Modified", "Removed") under which the relevant manual page can be found.

This is followed by a further alphabetical overview with brief comments.

Many of the modifications or corrections listed under the category "Modified" were already valid for Release Reliant UNIX 5.45A or earlier.

Simple corrections of printing or formatting errors are not included.

Tabular overview

Name	Modified	New	Removed
add_application 1M		X	
at 1	X		
bstring 3C	X		
catman 1M	X		
clone 3C		X	
crash 1M	X		
ctime 3C	X		
digi 7		X	
directory 3C	X		
errlogdo 1M		X	
errlogget 1M		X	

errlogreplace 1M		X
flockfile 3T		X
flup 8	X	
ftime 3C	X	
ftp 1	X	
ftpd 1M	X	
fuser 1M	X	
getc_unlocked 3T		X
getdtablesize 3C	X	
getgrent 3C	X	
gethostid 3N	X	
getlogin 3C	X	
getpriority 3C	X	
getpwent 3C	X	
getwd 3C	X	
index 3C	X	
killpg 3C	X	
log3 5	X	
log3adm 1M		X
log3admin 4	X	

log3do 1M		X
log3eval 1M		X
log3get 1M		X
log3header 5	X	
log3logger 1M	X	
log3open 5	X	
log3replace 1M		X
log3status 4		X
log3str 1M		X
mkstemp 3C	X	
mount 1M-devfs	X	
ppd 1M	X	
ps 1	X	
pthread 5		X
pthread_atfork 3T		X
pthread_attr_getguardsize 3T		X
pthread_attr_init 3T		X
pthread_attr_setdetachstate 3T		X
pthread_attr_setinheritsched(3T)		X

pthread_attr_setschedparam 3T	X
pthread_attr_setschedpolicy 3T	X
pthread_attr_setscope 3T	X
pthread_attr_setstackaddr 3T	X
pthread_attr_setstacksize(3T)	X
pthread_cancel(3T)	X
pthread_cleanup_push(3T)	X
pthread_condattr_getpshared 3T	X
pthread_condattr_init 3T	X
pthread_cond_init(3T)	X
pthread_cond_signal(3T)	X
pthread_cond_wait(3T)	X
pthread_create(3T)	X
pthread_detach(3T)	X
pthread_equal(3T)	X
pthread_exit(3T)	X
pthread_getconcurrency 3T	X
pthread_getschedparam 3T	X
pthread_intro 3T	X
pthread_join(3T)	X

pthread_key_create 3T	X
pthread_key_delete 3T	X
pthread_kill 3T	X
pthread_mutexattr_getpshared 3T	X
pthread_mutexattr_init 3T	X
pthread_mutexattr_setprioceiling 3T	X
pthread_mutexattr_setprotocol 3T	X
pthread_mutexattr_settype 3T	X
pthread_mutex_init(3T)	X
pthread_mutex_lock(3T)	X
pthread_mutex_setprioceiling 3T	X
pthread_once(3T)	X
pthread_rwlockattr_getpshared 3T	X
pthread_rwlockattr_init 3T	X
pthread_rwlock_init 3T	X
pthread_rwlock_rdlock 3T	X
pthread_rwlock_unloc	X

k 3T		
pthread_rwlock_wrloc k 3T		X
pthread_self(3T)		X
pthread_setcancelstat e 3T		X
pthread_setspecific(3 T)		X
queuedefs 4	X	
rand 3C	X	
random 3C	X	
regex 3C	X	
sar 1M	X	
sched 5		X
sched_yield 3T		X
sdb 1M		X
sdb.def 4		X
sgen 7	X	
sigprocmask 2	X	
sigwait(3T)		X
srout 7	X	
scp 1		X
slogin 1		X

ssh 1		X
ssh-add 1		X
ssh-agent 1		X
ssh-keygen 1		X
sshd 8		X
stape 7	X	
string 3C	X	
tcpdump 1M	X	
ttyname 3C	X	
utimes 3C	X	
vdadmin 8	X	
vdisklite 7	X	
wait3 3C	X	
yppasswd 1M	X	
zx 5	X	

Fujitsu Siemens Computers Add-on Reference Manual

man-news(1)

NAME

`man-news` - new or updated manual pages with Release 5.45B

DESCRIPTION

Three changes have been made to manual pages between Release Reliant UNIX 5.45A and Release Reliant UNIX 5.45B:

- New manual pages have been included
- Some manual pages have been removed
- Some manual pages have been modified or corrected

The following alphabetically ordered table indicates the particular category ("New", "Modified", "Removed") under which the relevant manual page can be found.

This is followed by a further alphabetical overview with brief comments.

Many of the modifications or corrections listed under the category "Modified" were already valid for Release Reliant UNIX 5.45A or earlier.

Simple corrections of printing or formatting errors are not included.

Tabular overview

Overview with brief comments

Name Comment

`add_application`(1M)
: Add a third-party Web application to WebSysAdmin

`at`(1)
: Added to `-q` option: `d` to `z` for user-defined queues

`bstring`(3C)
: Functions now included in `attlibc`

`catman`(1M)
: Description of `-m` option added

`clone`(3C)
: Create a child process

`crash`(1M)
: Note added to the `ep_stop` command

`ctime`(3C)
: `ctime_r()`, `localtime_r()`, `gmtime_r()`, and `asctime_r()` functions added (Pthread)

`digi`(7)
: Serial interfaces offered by AccelePort *Xr* 920 PCI boards

`directory`(3C)
: `readdir_r()` function added (Pthread)

`errlogdo`(1M)
: `DO` command for TMS (Threshold Monitoring System)

`errlogget`(1M)
: `GET` command for TMS (Threshold Monitoring System)

`errlogreplace`(1M)
: `SET` command for TMS (Threshold Monitoring System)

`flockfile`(3T)
: Standard input/output locking functions (UNIX 98)

`flup`(8)
: Completely new description

`ftime`(3C)
: Function now included in `attlibc`

`ftp`(1)
: No EBCDIC file transfer

`ftpd`(1M)
: Better description of an anonymous FTP user

`fuser`(1M)
: New *–signal* option and modified `–k` option

`getc_unlocked`(3T)
: Standard input/output with explicit client locking (UNIX 98)

`getdtablesize`(3C)
: Function now included in `attlibc`

`getgrent`(3C)
: `getgrgid_r()` and `getgrnam_r()` functions added (Pthread)

`gethostid`(3N)
: Function now included in `attlibc`

`getlogin`(3C)
: `getlogin_r()` function added (Pthread)

`getpriority`(3C)
: Functions now included in `attlibc`

`getpwent`(3C)
: `getpwnam_r()` and `getpwuid_r()` functions added (Pthread)

`getwd`(3C)
: Function now included in `attlibc`

`index`(3C)
: Functions now included in `attlibc`

`killpg`(3C)
: Function now included in `attlibc`

`log3`(5)
: General revision

`log3adm`(1M)
: Command for converting the `log3admin` administration file for Logging V3.0

`log3admin`(4)
: General revision

`log3do`(1M)
: `DO` command for Logging V3.0

`log3eval`(1M)
: Logbook evaluation (unformatted)

`log3get`(1M)
: `GET` command for Logging V3.0

`log3header`(5)
: General revision

`log3logger`(1M)
: General revision

`log3open`(5)
: General revision

`log3replace`(1M)
: `SET` command for Logging V3.0

`log3status`(4)
: Runtime status of Logging V3.0

`log3str`(1M)
: Command for converting the `log3struct` administration file for Logging V3.0

`mkstemp`(3C)
: Function now included in `attlibc`

`mount`(1M-devfs)
: New `/fcdisk` keyword and new `-o` option (Fibre Channel)

`ppd`(1M)
: Completely new description

`ps`(1)
: New `-C` option

`pthread`(5)
: Threads (UNIX 98)

`pthread_atfork`(3T)
: Register fork handlers (UNIX 98)

`pthread_attr_getguardsize`(3T)
: Get or set the thread *guardsize* attribute (UNIX 98)

`pthread_attr_init`(3T)
: Initialise and destroy threads attribute object (UNIX 98)

`pthread_attr_setdetachstate`(3T)
: Set and get *detachstate* attribute (UNIX 98)

`pthread_attr_setinheritsched`(3T)
: Set and get `inheritsched` attribute (**REALTIME THREADS**) (UNIX 98)

`pthread_attr_setschedparam`(3T)
: Set and get `schedparam` attribute (UNIX 98)

`pthread_attr_setschedpolicy`(3T)
: Set and get `schedpolicy` attribute (**REALTIME THREADS**) (UNIX 98)

`pthread_attr_setscope`(3T)
: Set and get `contentionscope` attribute (**REALTIME THREADS**) (UNIX 98)

`pthread_attr_setstackaddr`(3T)
: Set and get *stackaddr* attribute (UNIX 98)

`pthread_attr_setstacksize`(3T)
: Set and get *stacksize* attribute (UNIX 98)

`pthread_cancel`(3T)
: Cancel execution of a thread (UNIX 98)

`pthread_cleanup_push`(3T)
: Establish cancellation handlers (UNIX 98)

`pthread_condattr_getpshared`(3T)
: Get and set the process-shared condition variable attributes (UNIX 98)

`pthread_condattr_init`(3T)
: Initialise and destroy condition variable attributes object (UNIX 98)

`pthread_cond_init`(3T)
: Initialise and destroy condition variables (UNIX 98)

`pthread_cond_signal`(3T)
: Signal or broadcast a condition (UNIX 98)

`pthread_cond_wait`(3T)
: Wait on a condition (UNIX 98)

`pthread_create`(3T)
: Thread creation (UNIX 98)

`pthread_detach`(3T)
: Detach a thread (UNIX 98)

`pthread_equal`(3T)
: Compare thread IDs (UNIX 98)

`pthread_exit`(3T)
: Thread termination (UNIX 98)

`pthread_getconcurrency`(3T)
: Get or set level of concurrency (UNIX 98)

`pthread_getschedparam`(3T)
: Dynamic thread scheduling parameters access (**REALTIME THREADS**) (UNIX 98)

`pthread_intro`(3T)
: Introduction to UNIX 98 threads

`pthread_join`(3T)
: Wait for thread termination (UNIX 98)

`pthread_key_create`(3T)
: Thread-specific data key creation (UNIX 98)

`pthread_key_delete`(3T)
: Thread-specific data key deletion (UNIX 98)

`pthread_kill`(3T)
: Send a signal to a thread (UNIX 98)

`pthread_mutexattr_getpshared`(3T)
: Set and get process-shared attribute (UNIX 98)

`pthread_mutexattr_init`(3T)
: Initialise and destroy mutex attributes object (UNIX 98)

`pthread_mutexattr_setprioceiling`(3T)
: Set and get `prioceiling` attribute of mutex attribute object (**REALTIME THREADS**) (UNIX 98)

`pthread_mutexattr_setprotocol`(3T)
: Set and get protocol attribute of mutex attribute object (**REALTIME THREADS**) (UNIX 98)

`pthread_mutexattr_settype`(3T)
: Get or set a mutex type (UNIX 98)

`pthread_mutex_init`(3T)
: Initialise or destroy a mutex (UNIX 98)

`pthread_mutex_lock`(3T)
: Lock and unlock a mutex (UNIX 98)

`pthread_mutex_setprioceiling`(3T)
: Change the priority ceiling of a mutex (**REALTIME THREADS**) (UNIX 98)

`pthread_once`(3T)
: Dynamic package initialisation (UNIX 98)

`pthread_rwlockattr_getpshared`(3T)
: Get and set process-shared attribute of read-write lock attributes object (UNIX 98)

`pthread_rwlockattr_init`(3T)
: Initialise and destroy read-write lock attributes object (UNIX 98)

`pthread_rwlock_init`(3T)
: Initialise or destroy a read-write lock object (UNIX 98)

`pthread_rwlock_rdlock`(3T)
: Lock a read-write lock object for reading (UNIX 98)

`pthread_rwlock_unlock`(3T)
: Unlock a read-write lock object (UNIX 98)

`pthread_rwlock_wrlock`(3T)
: Lock a read-write lock object for writing (UNIX 98)

`pthread_self`(3T)
: Get calling thread's ID (UNIX 98)

`pthread_setcancelstate`(3T)
: Set cancelability state (UNIX 98)

`pthread_setspecific`(3T)
: Thread-specific data management (UNIX 98)

`queuedefs`(4)
: Added: `d` to `z` for user-defined queues

`rand`(3C)
: `rand_r()` function added (Pthread)

`random`(3C)
: Functions now included in `attlibc`

`regex`(3C)
: Functions now included in `attlibc`

`sar`(1M)
: `struct kmeminfo km;` --> `struct kmeminfo kmi[NCPU];` (from 5.44B)

  Note: The old `struct kmeminfo km;` (SINIX-N only) definition still applies for single-processor systems.

`sched`(5)
: Execution scheduling (**REALTIME THREADS**) (UNIX 98)

`sched_yield`(3T)
: Yield processor (UNIX 98)

`sdb`(1M)
: Tools for accessing the `sdb.daps` software database

`sdb.def`(4)
: Definition file for the `sdb.daps` software database

`sgen`(7)
: Description of SCC Addressing, Indexed Addressing and Auto Assigned Addressing added

`sigprocmask`(2)
: `pthread_sigmask()` function added (Pthread)

`sigwait`(3T)
: Wait for queued signals (UNIX 98)

`srout`(7)
: Description of SCC Addressing, Indexed Addressing and Auto Assigned Addressing added

`scp`(1)
: Secure copy (remote file copy program)

`slogin`(1)
: Secure login (remote login program)

`ssh`(1)
: Secure shell client (remote login program)

`ssh-add`(1)
: Adds identities for the authentication agent

`ssh-agent`(1)
: Authentication agent

`ssh-keygen`(1)
: Authentication key generation

`sshd`(8)
: Secure shell daemon

`stape`(7)
: Description of SCC Addressing, Indexed Addressing and Auto Assigned Addressing added

`string`(3C)
: `strtok_r()` function added (Pthread)

`tcpdump`(1M)
: Completely new description

`ttyname`(3C)
: `ttyname_r()` function added (Pthread)

`utimes`(3C)
: Function now included in `attlibc`

`vdadmin`(8)
: New options `-B`, `-u`, `-c select`, and `-c base2select`

`vdisklite`(7)
: New vdisk type `select`

`wait3`(3C)
: Function now included in `attlibc`

`yppasswd`(1M)
: New `-root` option

`zx`(5)
: New generic description

SEE ALSO

`manprint`(1).

Fujitsu Siemens Computers Add-on Reference Manual

mipcstat(1)

NAME

`mipcstat -` report MIPC statistics

SYNOPSIS

`mipcstat` [`–a`]

`mipcstat` [`–e`] [`–n` *nodename*] [*interval*] [*count*]

`mipcstat` [`–s`] [`–n` *nodename*] [*interval*] [*count*]

`mipcstat` [`–S`]

`mipcstat` [`–m`]

`mipcstat` [`–z`]

DESCRIPTION

`mipcstat` displays MIPC statistics collected per node since the last time they were cleared.

`mipcstat` always defaults to the current node if the node name is not specified. To retrieve statistics for a remote node from other nodes within the cluster, a node name must be specified. The `–n` option is used to specify the node name.

MIPC statistics can be monitored by specifying an *interval* in seconds. A running display of statistics accumulated over each interval is produced. For example, `mipcstat 5` will print the accumulated statistics for the local node every five seconds. If a count is given, the statistics are repeated *count* times.

OPTIONS

`–a` Print MIPC statistics for all the nodes in the cluster.

`–n` *nodename*
MIPC statistics for node name *nodename*.

`–e` A snap shot of endpoint descriptors (EPD) is displayed. The snapshot shows the current states of MIPC endpoint descriptors.

`NSID` Name space id.

`PORT` Port number.

`HASH` Hash index for this endpoint descriptor.

`OMCNT`
Count of messages sent out.

`IMCNT`
Count of messages received.

`IMSGQ`
Number of unread messages.

`RMSG` Number of free receive buffers.

`UMSG` Number of receive buffers in use.

`SMSG` Number of send buffers in use.

`DMSG` Number of free send buffers.

`PID` Pid of process owning the endpoint descriptor.

`-m` A snap shot of mipc regions is displayed. The snapshot shows the current states of MIPC registered regions.

`ADDR:`
registered region user address.

NSID:
name space id.

PORT:
port number.

REGSZ:
registered size of regions.

LCKSZ:
real locked memory size of regions.

PBDQ:
queue length for prepared buffers in one region.

PID: pid of process owning the region.

−s Provides general MIPC statistics.

Send Messages sent.

Recv Messages received.

NewRecv
Messages received but not yet read.

No EPD
Messages dropped due to no corresponding endpoint.

No Rbuf
Messages dropped due to no available receive buffers.

Rmemlim
Memory limit reached on receive.

Smemlim
Memory limit reached on send.

Buflim
Memory allocation failure for MIPC buffer.

LockedMem
Number of MIPC pages currently locked in memory.

SndCnt
Number of MIPC messages on the fly (sent, but not acked)

BufCnt
Number of MIPC buffers currently allocated

−S Print statistics concerning the MIPC packet size.

−z Clear all statistics on local node.

nodeinfo(1)

NAME

`nodeinfo` - convert between node names and node numbers for ReliantUNIX Cluster systems

SYNOPSIS

`nodeinfo` *number*

`nodeinfo` *name*

DESCRIPTION

If `nodeinfo` is given a number argument, it will attempt to find the node name for the given *number*.

If the argument is not numeric, it will attempt to find out which node number matches the given *name*.

If the conversion is successful, `nodeinfo` prints the converted value to the standard output and exits with a zero (true) exit status; otherwise, nothing is printed, and a non-zero (false) exit status is set.

System Administrator Commands(1M)

CCfgCmd(1M) - ctrldump(1M)

CCfgCmd(1M)

NAME

`CCfgCmd` - send commands to the Cluster Configuration Daemon

SYNOPSIS

`CCfgCmd` *command* [*argument* ...]

DESCRIPTION

The `CCfgCmd` utility may be used to send commands to a running `CCfgD` process.

Only a priviledged user is allowed to send commands to `CCfgD`.

The utility connects to `CCfgD` via the external communication channel and sends all its arguments as a command to it.

After the command was sent to `CCfgD` the `CCfgCmd` utility receives the command status and result and prints it to STDOUT or - in case of an error - to STDERR.

COMMUNICATION WITH CCfgD

When `CCfgD` starts, it establishes a UNIX domain socket as its external communication channel. `CCfgCmd` makes a connection to this socket and sends a message containing the command and the arguments.

When `CCfgD` is not running the connection establishment will fail and `CCfgCmd` aborts with the error message

```
FATAL: cannot create XCC client
```

COMMANDS

The `CCfgCmd` utility does not know about commands recognized by `CCfgD`. It simply packs the arguments into a message and sends them over the communication channel. However, the special command `CCfgCmd help` may be used to get a list of supported commands and a brief description. Please refer to `CCfgD`(1M) for further information.

SEE ALSO

`CCfgD`(1M) .

Fujitsu Siemens Computers Add-on Reference Manual

CCfgConfig(1M)

NAME

`CCfgConfig -` setup the configuration for the Cluster Config Daemon

SYNOPSIS

`CCfgConfig` [–c *config-file*] [–n *node-name*] *operation*

DESCRIPTION

The `CCfgConfig` utility may be used to set or get configuration information in the file `/etc/default/cluster` or in a similar file specified with the –c *config-file* option. See `cluster(4)` for more information on the format of this file.

OPTIONS

–c *config-file*

Use the file *config-file* instead of the default `/etc/default/cluster`.

–n *node-name*

Apply all operations to the configuration information of node *node-name*.

The default is to manipulate the configuration information for the special name `@node`, which denotes the local system.

OPERATIONS

`CCfgConfig` operates in one of four modes:

`get` *config-info* [*config-info* ...]

Retrieve the information *config-info* for the selected node and print it to standard output.

`set` *config-info value* [*config-info value* ...]

Configure the attribute *config-info* for the selected node to have the value *value*. A value of "–" indicates that the entry should be deleted.

If the indicated node has a configuration entry, the component(s) indicated by *config-info* is (are) modified.

If the config file does not contain an entry for the indicated node, a new entry for that node is created. In this case a minimum of information must be supplied:

`nodenum` *node-number*
`icf-`*ctlr0* `DLPI:`*device1*
`ip-addr` *ip-addr*

for an Ethernet Interconnect, or

`nodenum` *node-number*
`icf-`*ctlr0* `SCI:`*controller-number*
`ip-addr` *ip-addr*

for an SCI Interconnect.

It is not possible to delete these non-optional configuration entries unless the complete node configuration is deleted.

`print`

Print configuration information about all or the selected node to standard output.

`delete`

Delete the entire configuration information on all or the selected node.

INFORMATION ITEMS AND THEIR VALUES

The following table gives the description of the *config-info* keyword together with the exact input/output formats.

"`nodenum`"

`get` Print the node number.

Output format: *node-number*

An error message is printed to standard error when the configuration information does not contain a node number.

`set` Value input format: a decimal number in the range 1 ... 32.

"`nodename`"

`get` Print the node name.

Output format: *node-name*

Nothing is printed when the node name is not specified in the configuration file.

`set` Value input format: ASCII string, max. 31 characters long.

"`clustername`"

`get` Print the system name.

Output format: *system-name*

Nothing is printed when the system name is not specified in the configuration file.

`set` Value input format: ASCII string, max. 31 characters long.

"`icf`"

`get` Print the configuration information of ICF devices.

Output format: *ctlr net-type*:*device*[:*arguments*]

One line for each controller is printed. An error message is printed to standard error when the configuration information does not contain any ICF devices.

`set` Value input format: see "`icf-`*ctlr*".

"`icf-`*ctlr*"

`get` Print the configuration information of a particular ICF device with controller number *ctlr*.

Output format: *net-type*:*device*[:*arguments*]

An error message is printed to standard error when the configuration information does not contain the requested ICF device.

`set` Value input format: *net-type*:*device*[:*arguments*]

ctlr must be a decimal number in the range 0 ... 3. Currently, the *net-types* `DLPI` and `SCI` are supported, *device* must be a pathname to the physical device for the DLPI case and a SCI controller number for the SCI case. No additional *arguments* are required for the known net-types up to now.

"`ip`"

`get` Print the configuration information for the node's IPCF interface.

Output format: *ip-addr* [`netmask` *ip-netmask*] [`bcast`*ip-bcast-addr*]

An error message is printed to standard error when the configuration information does not contain a valid IP entry.

`set` Value input format: see "`ip-addr`", "`ip-netmask`" and "`ip-bcast-addr`".

"`ip-addr`"

`get` Print the IP address of the node's IPCF interface.

Output format: *ip-addr*

An error message is printed to standard error when the configuration information does not contain a valid IP address.

`set` Value input format: an IP address, eihter in dotted quad notation or a host name.

"`ip-netmask`"

`get` Print the IP netmask of the node's IPCF interface.

Output format: *ip-netmask*

Nothing is printed when the netmask is not specified in the IP configuration entry.

`set` Value input format: IP netmask, hexadecimal number, 8 hex digits.

"`ip-bcast-addr`"

`get` Print the IP broadcast address of the node's IPCF interface.

Output format: *ip-bcast-addr*

Nothing is printed when the netmask is not specified in the IP configuration entry.

`set` Value input format: an IP broadcast address in dotted quad notation.

EXAMPLES

```
# CCfgConfig -n alpha get nodenum
1
# CCfgConfig -n alpha get nodename clustername
cl-1
snicluster
# CCfgConfig -n alpha get icf-0 icf-1
DLPI:/dev/dlpi/zx1
DLPI:/dev/dlpi/zx2
# CCfgConfig -n alpha get nodenum icf ip
1
0 DLPI:/dev/dlpi/zx1
1 DLPI:/dev/dlpi/zx2
192.168.0.1 netmask 0xFFFFFF00
# CCfgConfig -n beta delete
# CCfgConfig -n beta print
ERROR: no configuration for node 'beta'
# CCfgConfig -n beta set nodenum 2 \
         nodename cl-2 \
         clustername snicluster \
         icf-0 DLPI:/dev/dlpi/zx1 \
         icf-1 DLPI:/dev/dlpi/zx2 \
         ip-addr cl-2 \
         ip-netmask 0xFFFFFF00
# CCfgConfig -n beta print
beta
nodenum 2
nodename cl-2
clustername snicluster
icf-0 DLPI:/dev/dlpi/zx1
icf-1 DLPI:/dev/dlpi/zx2
ip 192.168.0.2 netmask 0xFFFFFF00
```

SEE ALSO

`CCfgD`(1M), `cluster`(4).

Fujitsu Siemens Computers Add-on Reference Manual

CCfgD(1M)

NAME

CCfgD - Cluster Configuration Daemon

SYNOPSIS

CCfgD [–c *config-file*] [–no-daemon] [–d] [–df *features*] [–debug-file *file*] [–log-file *file*] [–t*timeout*] [–n *num*]

DESCRIPTION

CCfgD, the ICF Cluster Configuration Daemon, is normally run at boot time from a run-level 2 rc script. In normal operation mode, the program forks and works as a daemon program in background.

When started, CCfgD reads the cluster node configuration information from a config file, the default being /etc/default/cluster. See cluster(4) for more information on the format of this file. CCfgD parses the config file and extracts the cluster node configuration for the local node. It sets up the mapping of logical ICF devices to physical network interfaces and initializes the ICF devices. Then it starts the CCfgD node registration process. The purpose of this process is to make the local node information available to other cluster nodes and to gain information about the other nodes that are part of the cluster. When the node registration process is complete the local ICF is initialized and the system becomes part of the cluster.

After the node registration process is complete, CCfgD continues to listen on the cluster interconnect interfaces for node registration requests from other nodes that try to join the cluster.

Each node registration request is analyzed and answered with a node registration reply. Additionally CCfgD will update the local ICF when new nodes join the cluster.

NODE REGISTRATION PROCESS

CCfgD builds cluster node registration request packets and broadcasts them over the cluster interconnect, trying to contact CCfgD processes on other nodes of the cluster. Three node registration request packets are broadcasted on each physical network that is part of the interconnect, with some delay between each two packets.

CCfgD then waits 3 seconds for node registration replies from CCfgD processes on other cluster nodes. It collects the information from each reply packet and builds an internal cluster node table.

After the timeout of 3 seconds CCfgD initializes the ICF with information it got so far. After ICF initialization other ICF based services may start.

OPTIONS

–c *config-file*
: Read the configuration information from *config-file* instead of the default /etc/default/cluster.

–no-daemon
: Do not start in daemon mode, but operate as a normal process in foreground. In this mode it is possible to output diagnostic messages and log messages to STDOUT or STDERR (see sections **DEBUGGING** and **LOGGING**).

–d
: Enable basic diagnostic output.

–df *features*
: Enable extended diagnostic output. *features* is a logical OR of the following values:

  0x0001
  : Print information about sent/received packets.

  0x0002
  : Additionally dump packet data.

  0x0004

Print information about cluster node state changes.

`0x0008`
Print information about node id setting.

`0x0100, 0x0200, 0x0400`
Print various internal information for the SCI driver.

`–debug-file` *file*
Print diagnostic messages to *file* instead of the default `/var/adm/cluster/debug`.

`–log-file` *file*
Print log messages to *file* instead of logging them with `syslog(3C)`.

`–t` *timeout*
Set the node registration process timeout to *timeout* seconds. The default is 3 seconds.

`–n` *num*
Send *num* broadcast packets during the node registration process. The default is 3 packets.

LOGGING

`CCfgD` normally outputs log messages via `syslog(3C)`. It uses the facility `LOG_DAEMON`.

The following levels are used:

`LOG_INFO`
Informational messages.

`LOG_ERROR`
Error messages.

`LOG_ALERT`
Critical errors that need operator intervention.

It is also possible to write the log messages to a file instead of using `syslog(3C)`. This is the case when a log file name is given with `–log-file` *file*.

When `CCfgD` operates in non-daemon mode (option `–no-daemon`) it is possible to output log messages to STDOUT or STDERR:

`–log-file stdout`
Write log messages to STDOUT.

`–log-file stderr`
Write log messages to STDERR.

DEBUGGING

When diagnostic messages are enabled by either the `–d` or the `–df`*features* option, `CCfgD` writes such messages to a file. New messages are always appended to the end of the file. By default the file `/var/adm/cluster/debug` is used. The default file may be changed with the `–debug-file` *file* option.

When `CCfgD` operates in non-daemon mode (option `–no-daemon`) it is possible to output diagnostic messages to STDOUT or STDERR:

`–debug-file stdout`
Write diagnostic messages to STDOUT.

`–debug-file stderr`
Write diagnostic messages to STDERR.

PROGRAM STARTUP

Since `CCfgD` controls critical resources with respect to the ICF cluster interconnect, only one instance of `CCfgD` may be running at a time. `CCfgD` will first check at startup if another instance of the program is already running. If this is the case, the second attempt to start `CCfgD` will fail with the error message

```
FATAL: another instance of CCfgD is running
```

EXTERNAL COMMUNICATION CHANNEL

To allow some control of the `CCfgD` operation an external communication channel is installed. This is the UNIX domain socket `/var/adm/cluster/CCfgD.XCC`.

The program CCfgCmd may be used to send commands to `CCfgD`. The basic syntax is:

`CCfgCmd` *command* [*argument* ...]

Currently the following commands are implemented:

`help` Print a list of supported commands and a brief help text.

`exit` Causes `CCfgD` to cleanup and exit normally.

`debug` { `on` | `features` *f* | `off` }
Change the level for printing diagnostic messages.

`on` Enable debug output.

`features` *f*
Enable extended debug output for features *f*. See the option `-df` for a list of supported debug features.

`off` Disable debug output.

`node` *num* { `info` | `cinfo` }
Print information about a node *num*.

`info` Print node information from the `CCfgD` internal data.

`cinfo`
Print node information as retreived from ICF.

SIGNAL HANDLING

`CCfgD` handles the following signals:

`SIGTERM`
Cleanup and exit normally.

`SIGINT`
Ignored.

`SIGHUP`
Ignored.

`SIGQUIT`
Ignored.

All other signals are left to their default action.

FILES

`/etc/default/cluster`
`/var/adm/cluster/CCfgD.pid`
`/var/adm/cluster/CCfgD.XCC`
`/var/adm/cluster/debug`

SEE ALSO

`CCfgCmd`(1M), `syslog`(3C), `cluster`(4).

Fujitsu Siemens Computers Add-on Reference Manual

cddevsuppl(1M)

NAME

`cddevsuppl` - set and get the major and minor numbers of a devicefile

SYNOPSIS

`cddevsuppl` [–m *mapfile* | –u *unmapfile*] [–c]

`cddevsuppl` *mount-point*

DESCRIPTION

This command is used to map and unmap the major and minor numbers of a device file on a mounted CD-ROM.

If `cddevsuppl` is executed without the –m or –u option but with the mount-point argument, it lists the current device file mappings on the system.

The –m *mapfile* and –u *unmapfile* options are mutually exclusive.

OPTIONS

–m *mapfile*

This option will map the major and minor numbers for device files. The mappings are specified in *mapfile*. This file has one entry for each device file mapping in the format:

device_file_path new_major new_minor

The fields are separated by white space. The entries are separated by newlines. Anything beyond the third field shall be treated as a comment.

The maximum number of mappings is defined in the header file `<sys/cdrom.h>`. A previous device file mapping for a specific device file is overridden if that device file that is mapped again.

–u *unmapfile*

This option will unmap the major and minor numbers for device files. The mappings are specified in *unmapfile*. This file has one entry for each device file mapping in the format:

device_file_path

The entries are separated by newlines. Anything beyond the first field shall be treated as a comment.

–c This option is only useful when used in combination with the –m*mapfile* or –u *unmapfile* options. The –c option will cause `cddevsuppl` to continue processing the device file mappings if an error is returned for a specific device file mapping. An error message for that specific device file will be printed to standard error. The default action is to stop processing when an error has occurred.

STDOUT

If no options are used the current device file mappings are listed on standard output. In the case of –m *mapfile*, the new setting is listed if the mapping was completed successfully. In the case of –u*unmapfile*, the device file and the major/minor numbers as recorded on the CD-ROM are listed if the unmapping was completed successfully.

EXIT STATUS

The following exit values are returned:

`0` Successful completion.

`1` File not found or file is not a file or directory within a CD-ROM file hierarchy or access denied.

`2` Not user with appropriate privileges.

`3` Too many mappings.

`4` Parameter error or bad format in a mapping file.

5 File is not a device file.

6 File not previously mapped.

APPLICATION USAGE

Only a user with appropriate privileges may change administrative CD-ROM features successfully. To read the current device file mappings, the user must have read permission on the device file.

Mappings should be established before affected device files are used. If the command is applied for device file mappings when device files have already been opened, the effect of this command on these files is undefined.

The device file mappings for a mounted CD-ROM are eliminated when the CD-ROM is unmounted.

ENVIRONMENT VARIABLES

No environment variables affect the execution of `cddevsuppl`. Note that `LC_TYPE` will not be used in file name conversion.

Fujitsu Siemens Computers Add-on Reference Manual

cddrec(1M)

NAME

`cddrec -` read Directory Record from CD-ROM directory

SYNOPSIS

`cddrec` [`–s` *number*] [`–b`] *file*

DESCRIPTION

This command is used to access the Directory Record associated with a CD-ROM file or directory and to list its contents on standard output.

If the command is used without the option `–b`, only the fixed part of the Directory Record is copied from CD-ROM; this does not include the System Use field.

OPTIONS

`–s` *number*
: This option specifies the File Section for which the Directory Record should be read. The numbering starts with one. If this option is omitted the last File Section of that file is assumed.

`–b`
: With this option the entirety of Directory Record is copied from the CD-ROM to standard output in binary format.

OPERANDS

The operand *file* is the name of any file or directory within the CD-ROM file hierarchy.

STDOUT

The output is formatted in the form of a table which contains the name of each field of the Directory Record and the corresponding contents of the entry as recorded on the CD-ROM. The System Use field is not listed because it may contain non-printable characters. If the option `–b` is applied, the contents of the Directory Record is written to the standard output in binary format as it is on the CD-ROM. The System Use field is included.

EXIT STATUS

The following exit values are returned:

`0`
: Successful completion.

`1`
: *file* not found or *file* is not within a CD-ROM file hierarchy, or access permission denied.

`2`
: File Section indicated by `–s` does not exist.

NOTES

The user must have read permission for *file* to execute the command successfully.

ENVIRONMENT VARIABLES

`LC_TIME` determines the format and contents of date and time strings. If `LC_TIME` is not set in the environment or is set to the empty string, the value of `LANG` will be used as a default. If `LANG` is not set or set to the empty string, the corresponding value from the implementation-specific default locale will be used. If `LC_TIME` or `LANG` contain an invalid setting, the utility will behave as if none of the variables had been defined.

Fujitsu Siemens Computers Add-on Reference Manual

cdmntsuppl(1M)

NAME

`cdmntsuppl` - set and get administrative CD-ROM features

SYNOPSIS

`cdmntsuppl` [`–u` *owner*] [`–g` *group*] [`–F`*mode*] [`–D` *mode*] [`–U` *umfile*] [`–G` *gmfile*] [`–c`] [`–l`] [`–m`] [`–x`] [`–s`] *mount-point*

DESCRIPTION

This command sets up administrative CD-ROM features, such as default ownership and access permissions, mapping of user and group identifications, conversion of file names and setting of execute permissions for directories. Each combination of options may be used, including no option at all. If `cdmntsuppl` is executed without any option, it lists the current setting.

OPTIONS

The options `–u`, `–g`, `–F` and `–D` may be used to set the default owner, group and/or default access permissions to be associated with those files and directories in a CD-ROM file system that do have a restricted final XAR, or no final XAR. This is useful in situations where files are supplied with a restricted final XAR, or no final XAR, but access specific to one user or group (other than the default at mount time) is required, or where the default access permissions (being read and execute permission for user, group and others) for files and directories are inappropriate.

`–u` *owner*

The operand *owner* may be either a decimal user ID or a login name found in the User Database.

`–g` *group*

The operand *group* may be either a decimal group ID or a group name found in the Group Database.

`–F` *mode*

This option is used to set the default permissions for files. The permissions are changed according to *mode*, which may be absolute or symbolic. An absolute mode is a four-octal-digit number constructed from the logical-or (sum) of the following modes:

`0400` read by owner

`0100` execute by owner

`0040` read by group

`0010` execute by group

`0004` read by others

`0001` execute by others

A symbolic mode has the form:

[*who*] *op* [*permission*]

The *who* part is a combination of the letters `u` (user), `g` (group) and `o` (other). The letter `a` stands for `ugo`, the default if *who* is omitted.

The argument *op* can be `+`, to add permission to the file's mode, `–`, to take away *permission*, or `=`, to assign *permission* absolutely (all other bits will be reset).

The argument *permission* is any combination of the letters `r` (read) and `x` (execute); `u`, `g` or `o` indicate that *permission* is to be taken from the current mode. Omitting *permission* is only useful with `=` to take away all permissions.

`–D` *mode*

This option is used to set the default permissions for directories in the same fashion as for the option `–F`. Execute permission will be interpreted as search permission.

The options –U and –G may be used in the case where a CD-ROM file system has been supplied with undesirable User/Group IDs associated with files and directories. These undesirable IDs may be transformed to more appropriate numbers. Only files and directories with an unrestricted final XAR are subject to this mapping (see [–u *owner*] and [–g *group*] above for other files and directories). An owner or group identification of zero is not permitted by ISO 9660 to appear in an unrestricted XAR, and thus the behaviour is undefined if a mapping is given for the value zero.

–U *umfile*, –G *gmfile*

These options need a file (*umfile*/*gmfile*) as operand which must contain pairs. Each pair must be provided using the following syntax: value as on CD-ROM, colon, numeric ID or User/Group name as found in User/Group Database. Pairs must be separated by a newline character. The maximum number of mappings is defined in the header file <sys/cdrom.h>.

The –c, –l and –m options establish/de-establish a specific name conversion of File/Directory Identifiers on a CD-ROM. Name conversion is a desirable feature to represent File/Directory Identifiers in a way that is in accordance to common conjunction. The option –c may not be used in conjunction with the options –l or –m.

–c This option causes names to be handled as recorded on CD-ROM, i.e. no conversion takes place. This matches the default after initialisation.

–l Upper case characters in Identifiers are converted to lower case. If the File Identifier contains no File Name Extension, the SEPARATOR 1 (.) is not represented. The effect of the –m option is unchanged.

–m The Version Number and the SEPARATOR 2 (;) of a File Identifier is not represented. The effect of the –l option is unchanged.

The –x and –s options determine the setting of the execute (search) permission bits for those directories in the CD-ROM file hierarchy that have a non-restricted final XAR.

–x The execute permission bits for directories within the CD-ROM file hierarchy are set as provided in the permissions field in the XAR of that directory.

–s Each execute permission bit for a directory in the XSI file Hierarchy is set to the inclusive OR of the corresponding read and execute bits in the XAR of that directory on the CD-ROM.

OPERANDS

The operand mount-point is the name of the mount-point of the CD-ROM file system.

STDOUT

If no options are used the current settings are listed on the standard output. In the case of setting features the new setting is listed if the command is completed successfully.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

1 Mount-point not found, or is not mount-point of a CD-ROM file system, or access permission denied.

2 Not user with appropriate privilege.

3 Too many mappings.

4 Parameter error or bad format in a mapping file (*umfile*/*gmfile*)

APPLICATION USAGE

Only a user with appropriate privileges may change administrative CD-ROM features successfully. To read the current settings the user must have read permission on the mount-point of that CD-ROM file system. In case of setting CD-ROM features this command is intended to be used only directly after mounting the CD-ROM and before any access to the CD-ROM is done. If the command is applied for setting features when files or directories have already been opened the effect of this command on these files and directories is undefined.

ENVIRONMENT VARIABLES

No environment variables affect the execution of `cdmntsuppl`. Note that `LC_TYPE` will not be used in file name conversion.

Fujitsu Siemens Computers Add-on Reference Manual

cdptrec(1M)

NAME

`cdptrec -` read Path Table Record from CD-ROM Path Table

SYNOPSIS

`cdptrec` [`–b`] *directory*

DESCRIPTION

This command is used to access a Path Table Record associated with a CD-ROM directory and lists its contents on the standard output.

OPTIONS

`–b` With this option the Path Table Record is copied from CD-ROM to standard output in binary format.

OPERANDS

The operand *directory* is the name of any directory within the CD-ROM file hierarchy.

STDOUT

The output is formatted in the form of a table which contains the name of each field of the Path Table Record and the corresponding contents of the entry as recorded on the CD-ROM.

If the `–b` option is applied, the contents of the Path Table Record is copied to the standard output in binary format as it is on the CD-ROM.

EXIT STATUS

The following exit values are returned:

`0` Successful completion.

`1` *directory* not found or is not within a CD-ROM file hierarchy, or access permission denied.

NOTES

The user must have read permission for *directory* to execute the command successfully.

ENVIRONMENT VARIABLES

No environment variables affect the execution of `cdptrec`.

Fujitsu Siemens Computers Add-on Reference Manual

cdsuf(1M)

NAME

`cdsuf -` read the System Use Fields from a System Use Area

SYNOPSIS

`cdsuf` [`–s` *number*] [`–b`] *file*

DESCRIPTION

This command is used to access the System Use Fields of the System Use Area associated with a File Section of a file or directory and to list its contents on standard output, following any Continuation Fields that may be present.

OPTIONS

`–s` *number*
: This option specifies the File Section for which the System Use Area shall be read. The numbering starts with one. If this option is omitted the last File Section of that file is assumed.

`–b`
: With this option all of the System Use Fields of the System Use Area are copied from the CD-ROM to standard output in binary format.

OPERAND

The operand *file* is the name of any file or directory within the CD-ROM file hierarchy.

STDOUT

The output is formatted in the form of a table which contains an entry for each System Use Field in the System Use Area as recorded on the CD-ROM. Each entry of the table shall have the fields *Signature*, *Length*, *Version*, and *Data* as specified in the System Use Sharing Protocol. Whether to break up the *Data* field into smaller fields according to the protocol which defined the *Signature* field is left up to the implementation.

If the `–b` option is applied, the contents of the full System Use Area are written to standard output in binary format as it is recorded on the CD-ROM.

EXIT STATUS

The following exit values are returned:

`0` Successful completion.

`1` File not found or file is not a file or directory within a CD-ROM file hierarchy or access denied.

`2` File Section indicated by `–s` does not exist.

`3` File Section indicated by `–s` has no System Use Area.

NOTES

The user must have read permission for *file* to execute the command successfully.

ENVIRONMENT VARIABLES

`LC_TIME` determines the format and contents of date and time strings. If `LC_TIME` is not set in the environment or is set to the empty string, the value of `LANG` will be used as a default. If `LANG` is not set or set to the empty string, the corresponding value from the implementation-specific default locale will be used. If `LC_TIME` or `LANG` contain an invalid setting, the utility will behave as if none of the variables had been defined.

Fujitsu Siemens Computers Add-on Reference Manual

cdvd(1M)

NAME

`cdvd -` read Volume Descriptor from CD-ROM

SYNOPSIS

`cdvd [ –b ]` *file*

DESCRIPTION

This command is used to read the Primary Volume Descriptor from a CD-ROM and to list the contents on the standard output.

OPTION

`–b` With this option the entirety of the Primary Volume Descriptor is copied from CD-ROM to standard output in binary format.

OPERANDS

The operand *file* is either a file or directory within the CD-ROM file hierarchy of the mounted CD-ROM file system, or the name of the block special file for a CD-ROM file system.

STDOUT

The output is formatted in the form of a table which contains the name of each field of the Primary Volume Descriptor and the corresponding contents of the entry as recorded on the CD-ROM. The Application Use field is not listed because it may contain non-printable characters.

If the `–b` option is applied, the contents of the Primary Volume Descriptor is copied to the standard output in binary format as it is on the CD-ROM. This includes the Application Use field.

EXIT STATUS

The following exit values are returned:

`0` Successful completion.

`1` *file* not found or *file* is not within a CD-ROM file hierarchy, or *file* is not a block special file for a CD-ROM file system, or access permission denied.

`2` *file* is a block special file and a read error occurred, or the CD-ROM is not recorded according to the ISO 9660 standard.

NOTES

The user must have read permission for *file* to execute the command successfully.

FUTURE DIRECTIONS

This command may be extended for further types of Volume Descriptors (e.g. Supplementary Volume Descriptor, Volume Partition Descriptor). For the purpose additional options will be introduced.

ENVIRONMENT VARIABLES

`LC_TIME` determines the format and contents of date and time strings. If `LC_TIME` is not set in the environment or is set to the empty string, the value of `LANG` will be used as a default. If `LANG` is not set or set to the empty string, the corresponding value from the implementation-specific default locale will be used. If `LC_TIME` or `LANG` contain an invalid setting, the utility will behave as if none of the variables had been defined.

Fujitsu Siemens Computers Add-on Reference Manual

cdxar(1M)

NAME

`cdxar -` read Extended Attribute Record from CD-ROM

SYNOPSIS

`cdxar` [`–s` *number*] [`–b`] *file*

DESCRIPTION

This command is used to access the Extended Attribute Record associated with a File Section of a file or directory and to list its contents on the standard output. If the command is used without the option `–b`, only the fixed part of the XAR is copied from the CD-ROM; this does not include the Application Use field and the Escape Sequences field.

OPTION

`–s` *number*
: This option specifies the File Section for which the XAR shall be read. The numbering starts with one. If this option is omitted the last File Section of that file is assumed.

`–b` With this option the entirety of XAR is copied from the CD-ROM to standard output in binary format.

OPERANDS

The operand *file* is the name of any file or directory within the CD-ROM file hierarchy.

STDOUT

The output is formatted in the form of a table which contains the name of each field of the XAR and the corresponding contents of the entry as recorded on the CD-ROM. The Application Use field and the Escape Sequences are not listed because they may contain non-printable characters.

If the option `–b` is applied, the contents of the full XAR is written to the standard output in binary format as it is on the CD-ROM.

EXIT STATUS

The following exit values are returned:

`0` Successful completion.

`1` *file* not found, or *file* is not file or directory within a CD-ROM file hierarchy, or access permission denied.

`2` File Section indicated by `–s` does not exist.

`3` File Section indicated by `–s` has no XAR.

`4` The File Section of *file* indicated by `–s` is not on the currently mounted CD-ROM.

NOTES

The user must have read permission for *file* to execute the command successfully.

ENVIRONMENT VARIABLES

`LC_TIME` determines the format and contents of date and time strings. If `LC_TIME` is not set in the environment or is set to the empty string, the value of `LANG` will be used as a default. If `LANG` is not set or set to the empty string, the corresponding value from the implementation-specific default locale will be used. If `LC_TIME` or `LANG` contain an invalid setting, the utility will behave as if none of the variables had been defined.

Fujitsu Siemens Computers Add-on Reference Manual

clos_adm(1M)
Cluster Operating System

NAME

clos_adm - managing CLOS cluster names

SYNOPSIS

clos_adm –A *clos_name* [–S *name*]

clos_adm –L

clos_adm –R *clos_name* [–S *name*]

clos_adm –S *name*

clos_adm –?

DESCRIPTION

In the context of CLOS, we use the term cluster to refer to a set of UNIX systems (nodes) which form an administration unit and communicate with one another via a communications medium. Every cluster has its own distinctive name since any one system can comprise several such administration units. The cluster names are administered on the local node using the clos_adm command. Nodes are assigned to a cluster with the node_adm(1M) command.

COMMAND OPTIONS

–A (Add) – Adding a CLOS cluster

This command is used to configure a *clos_name* name for a cluster on the local node. The command creates the /var/clos/*clos_name* home directory and all necessary subdirectories for a cluster incarnation. An empty CLODB Cluster Object Database is also installed [see clodb(7)] with the accompanying archive. A Closmon incarnation [see closmon(1M)], which still does not recognize any nodes, is then configured. Only after node_adm(1M) is subsequently called with the –A option are nodes added to the cluster called *clos_name*.

Note:

A node can only be added to a cluster if the corresponding cluster name has been configured on it. A network can have several clusters with the same cluster name. An attempt to add a node to two clusters with the same name is rejected when node_adm(1M) is called.

A *clos_name* argument must be specified.

–L (List) – Listing the clusters and their nodes

This command is used to list all the cluster names configured on the node together with the accompanying nodes and their states [see node_adm(1M) , –L option]. In addition to cluster names, the generated list also shows which cluster name is currently preset.

–R (Remove) – Removing a cluster name

Using this command, the *clos_name* cluster name is deleted on the local node. All configured nodes are first removed and the accompanying directories deleted [see node_adm(1M) , –R option]. The Closmon incarnation associated with *clos_name* is then deleted [see sacadm(1M) , –r option]. If the cluster name removed is preset for the node, a warning is issued.

A *clos_name* argument must be specified.

–S (Set) – Setting the presetting

This command is used to preset the cluster name to the *name* of the argument. This can occur following the Add or Remove commands or independently of both commands. The presetting is entered in the /var/clos/.CLOSdefault file.

A *name* argument must be specified.

–? Usage of this command.

DIAGNOSTICS

The `clos_adm` command terminates with the return value 0 if no errors occurred. Error messages are output to standard error output with the following return values:

1 "`use command option exclusively`". Incorrect call: Command options are exclusive.

2 "`CLOSNAME` *clos_name* `too large (max 14)`". Incorrect call: The name of a cluster may not exceed 14 characters.

3 "`you need to be superuser`". Incorrect call: The command may only be performed by a superuser.

4 "`Can't assign CLOS name`". Situation error: The name of the cluster could not be determined.

5 "`No such CLOS '`*clos_name*`' configured`". Situation error: There is no cluster with the specified name.

6 "`Can't create directories`". Situation error: The user cannot create the subdirectories for a cluster in `/var/clos`.

FILES

`/var/clos`

`/var/clos/.CLOSdefault`

`/var/clos/`*clos_name*

SEE ALSO

`closmon`(1M), `node_adm`(1M), `sacadm`(1M), `clodb`(7), `clos`(7).

Fujitsu Siemens Computers Add-on Reference Manual

clos_cmd(1M)
Cluster Operating System

NAME

clos_cmd - distributing a command in the cluster

SYNOPSIS

clos_cmd [–C *clos_name*] [–T] [–t *taglist*] –s *cmd_string*

clos_cmd [–C *clos_name*] [–T] [–t *taglist*] –n *cmd_path*

clos_cmd –?

DESCRIPTION

The clos_cmd command enables a CLOS subsystem [see clos(7)] to execute a command on all accessible nodes in the cluster. The command is executed asynchronously and in parallel on the nodes. There is no guarantee that execution of a command will yield the same result on all nodes. Error messages and return values from the command are ignored on all nodes. Similarly, there is no guarantee that a command will be executed within a defined period. Using the Closmon monitor [see closmon(1M)], clos_cmd distributes the command by datagram to the nodes, and the command is executed by the cluster service daemon clserver(1M) on the target node.

A check is carried out to determine whether the caller is known on the target node. If the caller is not known, the command is not executed. This is marked in the /var/saf/clos_service/log file on the relevant node. All commands are executed in the $HOME directory with the $HOME, $PATH and $USER environment variables.

OPTIONS

–C *clos_name*

The name of the cluster is selected in the *clos_name* argument using this option. If this option is not specified, the name from the $CLOSNAME environment variable is used, or if this is not set, the name is taken from the /var/clos/.CLOSdefault file.

–n *cmd_path*

Executing a command.

The *cmd_path* argument represents a binary file that can be executed under Reliant UNIX. Before this file is distributed, a check is performed locally to determine whether such an executable command exists. The file is distributed once again by closmon and clserver. On the target nodes, the $HOME, $PATH and $USER environment variables are converted to the caller values on the target node and are executed with exec(2) in the $HOME directory.

There is no controlling TTY for the program. Although standard input and standard output are closed, standard error output is redirected to the /var/saf/clos_service/cmd_err file.

–s *cmd_string*

Executing a shell command string.

The *cmd_string* argument represents a shell command string that can be executed under Reliant UNIX. This string is distributed via the Closmon monitor to all accessible nodes where it is executed by the clserver program. In order to execute the command, the caller's standard shell is started on every node in the respective $HOME directory with the corresponding rights. The $PATH, $HOME and $USER environment variables are taken from the /etc/passwd file of the respective node. The command return value is ignored.

–t *taglist*

Selecting the target node(s).

The *taglist* argument is a list of tag names for the target nodes on which the command is executed. A list consists of the tag names separated by commas. A white space character (blank, tab, newline

character) completes the list. If this option is not used, the command is executed on all active nodes. Otherwise, the command is only executed on the nodes set with the `-t` option. This option can be used several times. Invalid tag names are ignored.

If the `-T` option was used, the command is executed on all nodes *except* for those on the list.

`-T` Inverting the target node list.

The list of target nodes specified with the `-t` option is inverted. This means that the *cmd_string* command is executed on all nodes that are *not* on the list. If the list is blank (no `-t` option), the command is executed on all nodes except the one on which `clos_cmd` was called.

`-?` Usage of this command.

DIAGNOSTICS

If the `clos_cmd` command comes across an error situation locally before it is distributed, this is reported and the command is not executed on any node. If the `-n` option is used, `clos_cmd` checks whether the command can be executed locally before it is distributed. The return value is 0 if no error is discovered.

Since the command is sent by datagram, no responses are issued if errors occur when transferring the command or executing it on the remote nodes. If need be, an error message can be found in the `clserver` command log file.

`1` "`Illegal arguments`". Incorrect call.

`2` "`Can't use CLOS service! Closmon not running`". The local node is not integrated in the cluster (`NOT_RUNNING` or `NOT_AVAIL` state).

`3` "`Can't access _pmpipe!`" The caller cannot write to the `/etc/saf/`*clos_name*`/_pmpipe` file.

`4` "`Can't access pathname`". The command to be executed could not be found locally.

`5` "`You must use -s or -n option`". No command was specified.

FILES

`/etc/passwd`

`/etc/saf/`*clos_name*`/_pmpipe`

`/var/saf/clos_service/log`

`/var/saf/clos_service/cmd_err`

`/var/clos/.CLOSdefault`

SEE ALSO

`ksh`(1), `sh`(1), `closmon`(1M), `clserver`(1M), `exec`(2), `passwd`(4), `environ`(5).

Fujitsu Siemens Computers Add-on Reference Manual

clos_date(1M)
Cluster Operating System

NAME

clos_date - set date and time in the cluster

SYNOPSIS

clos_date [–C *clos_name*] [–a] [–T] [–t *taglist*] [–u] [*date_string*]

clos_date –?

DESCRIPTION

The clos_date command can be used by a CLOS subsystem [see clos(7)] to set the date and time on all specified and accessible nodes in the cluster. This procedure is synchronized and performed in parallel on the nodes. There is absolutely no guarantee given that this will happen within a predefined time. However, under normal circumstances it can be assumed that the time on the nodes may deviate by at most one second. The precision of the clocks can be checked with the timedc(1M) command. clos_date issues the date(1) command via the node channels [see clos(7)] and waits for acknowledgement.

The accessibility of the selected nodes is also checked. If a node is selected that cannot be accessed at present, the command terminates with errors if the –a option is not set. Only when all nodes are accessible or the A mode [see –a option] is enabled, does transmission begin. If a command could not be executed on a node or if the transmission to a node failed, a corresponding flag is set in the return value for the command.

If the *date_string* argument is not specified, the current date and time on the local node is established and distributed to all chosen nodes.

FORMATS

The following format is permitted for specifying the date in the *date_string* argument [cf. date(1)]:

[*mmdd*]*HHMM*[[*cc*]*yy*]

where:

mm is a two-position number for the month in the range 01 to 12. If this entry is not made, the current month is assumed.

dd is a two-position number for the day in the month in the range 01 to 31. If this entry is not made, the current day is assumed.

HH is a two-position number for the hour in the range 00 to 23.

MM is a two-position number for the minute in the range 00 to 59.

cc is a two-position number for the century –1, i.e. 19 or 20. If this entry is not made, the current century set is assumed.

yy is the last two digits of the century. If this entry is not made, the current year is assumed.

For *yy*, only the digits >69 and <38 may be entered. If the year number specified is <38, this is beyond the year 2000.

OPTIONS

–C *clos_name*

With this option, the name of the cluster is selected in the *clos_name* argument. If this option fails, the name from the $CLOSNAME environment variable is used or, if this is not set, the name is taken from the /var/clos/.CLOSdefault file.

–a Enabling A mode (do not abort).

In this mode, the command is not aborted if one of the selected nodes is not accessible. A suitable message is issued in this case for the node and the appropriate flag is set in the return value.

–t *taglist*

Selecting the target node(s).

The *taglist* argument is a list of tag names for the target nodes on which the date is set. A list comprises the tag names separated by commas. A "white space character" (blank, tabulator, newline character) terminates the list. If this option is not used, all configured nodes belong to the list.

The date is set on all nodes on the *taglist* list. If one of the nodes on the list cannot be accessed when the call is issued and if A mode is not enabled, `clos_date` terminates with errors.

If the –T option has been used, the date is set on all nodes except for those on the list.

–T Inverting the target node list.

The list of target nodes specified with the –t option is inverted. This means that the *date_string* date is executed on all nodes that are *not* on the list. If the list is blank (no –t option), the command is executed on all nodes except the one on which `clos_date` was called.

–u Using world time (UTC).

If this option is used, the date is used in accordance with the UTC (Universal Time Coordinated) world time, which corresponds to Greenwich Mean Time (GMT).

–? Usage of this command.

DIAGNOSTICS

If the `clos_date` command discovers an error situation locally before the `date(1)` command is issued, this is reported and the command is not executed on any of the nodes on the *taglist* list. The return value is 0 if no error is discovered. If an error is reported, the return value can be interpreted as follows: Return values are less than or equal to 8. No command has been transferred or started in these cases. A message is sent to standard error output and the return values have the following meaning:

1 "`illegal option`". Incorrect call.

2 "`local node not configured`". The local node is not configured in the cluster.

3 "`local node not integrated`". The local node is not integrated in the cluster (`NOT_READY` or `NOT_AVAIL` status).

4 "`Open channel to NODE` *tag* `fails!`". The caller cannot open the specified node channel.

5 "`Poll system call failed!`". The `poll(2)` system call failed.

6 "`not enough space for poll array`". The program has insufficient address space.

7 "`bad conversion`". The *date_string* argument has an invalid format

8 "`permission denied`". The calling user does not have the right permissions to set the date.

If the transfer and execution of the command has already commenced, confirmation is awaited from every node and sent to standard output. If an error occurred on one of the node channels when the command was being transferred or before it was executed, the acknowledgement text has the following format:

`NODE` *tag*`:` *acknowledgement_text*`! errno` *number* `->` *error_text*

The return value then contains the following values:

16 "`Can't send cmd request`". The command could not be issued on the node channel.

32 "`Wrong ID on CLOS_ACKN response`". `CLOS` did not process the commands correctly.

64 "`POLLERR on channel to NODE` *tag*". A `poll(2)` error occurred on one of the channels.

128 "`Command Transmission failed`". Transmission of the command failed on one of the channels.

FILES

`/var/clos/.CLOSdefault`

`/etc/passwd`

SEE ALSO

date(1), ksh(1), sh(1), closmon(1M), timedc(1M), poll(2), passwd(4), environ(5), clos(7).

Fujitsu Siemens Computers Add-on Reference Manual

clos_get(1M)
Cluster Operating System

NAME

`clos_get` - fetch file from a cluster node

SYNOPSIS

`clos_get` [`–C` *clos_name*] [`–z` *sec*] [`–i` *icmd*] [`–p` *pcmd*] [*sourcepath* [*destpath*]]

`clos_get –?`

DESCRIPTION

The `clos_get` command can be used to copy a *sourcepath* file to *destpath* on the local node from any of the nodes in the *clos_name* cluster. The transfer is not atomic, i.e. each cluster service daemon [see `clserver`(1M)] in the remote nodes is requested to send the file. The first node to acknowledge the request positively is termed the "active" remote node. It begins to transfer the file. If this fails, a different node that acknowledged the request becomes the "active" node and is requested to begin the transfer. When a transfer has succeeded, all acknowledged requests are aborted and the command terminates.

Configured nodes that were not accessible when the command was issued do not receive a copy request. If no node was accessible after a default time of 30 seconds or if all transfers were aborted, the command terminates with errors. The out time can be modified with the `–z` option.

If the *sourcepath* argument was not specified, the `COD` archive `/var/clos/`*clos_name*`/COD` [see `clodb`(7)] is copied from any node. If the *destpath* entry is not made, the pathname of *sourcepath* is assumed as the destination. Both arguments must be absolute pathnames.

OPTIONS

`–C` *clos_name*
: With this option, the name of the cluster is selected in the *clos_name* argument. If this option fails, the name from the `$CLOSNAME` environment variable is used or, if this is not set, the name is taken from the `/var/clos/.CLOSdefault` file.

`–i` *icmd*
: Execution of an *icmd* initial command.

  The pathname for an *icmd* command, which was executed on the "active" node prior to the transfer, is specified with the `–i` option. This command can be a script, which guarantees the consistency of the data during the transfer.

  A shell is generated on the "active" node with authorization for the calling user who executes the command.

`–p` *pcmd*
: Execution of a *pcmd* terminating command.

  The pathname for a *pcmd* command executed on the "active" node subsequent to the transfer is specified with the `–p` option.

  A shell with root authorization, which executes the command, is generated on the "active" node.

`–z` *sec*
: Modification of the default monitoring time.

  The default timeout is 30 seconds. This default time can be modified with this option. This means that a transfer request must have been initiated a maximum of *sec* seconds after the command was issued. Previously initiated transfers, which have not yet ended after *sec* seconds, are not aborted.

`–?` Usage of this command.

DIAGNOSTICS

The command terminates when a transfer ends, all transfers were aborted with errors or the timeout expires. If the file was copied successfully, the return value is 0. If an error occurs, the cause of the error is sent to standard output in the following format for each affected node:

`NODE` *tag*`:` *acknowledgement_text*`! errno` *number* `->` *error_text*

The return value then contains the following values:

`1` "`Illegal arguments`". Incorrect call.

`2` "`Can't assign CLOS name`". Configuration error or incorrect call: The `CLOS` name could not be established or no cluster of this name exists.

`3` "`Can't use sockets to communicate!`". Situation error: Communication via `socket`(3N) is not possible. The return value of `socket`(3N) provides more details.

`4` "`transmission error`". All file transfers from accessible nodes failed. The error codes of the individual transfers are reported to standard error output.

`5` "`timed out`". A transfer could not be initiated within the period of the timeout.

FILES

`/var/clos/.CLOSdefault`

`/var/clos/`*clos_name*`/COD`

SEE ALSO

`closmon`(1M), `clserver`(1M), `socket`(3N), `clodb`(7), `clos`(7).

Fujitsu Siemens Computers Add-on Reference Manual

clos_lock(1M)
Cluster Operating System

NAME

clos_lock - synchronously set a lock file in the cluster

SYNOPSIS

clos_lock [–C *clos_name*] [–a] [–T] [–t *taglist*] [–k *text*] [–v] [–w *sec*] –p *pathname*

clos_lock –?

DESCRIPTION

Using the clos_lock command, a CLOS subsystem [see clos(7)] can create a lock file on all specified and accessible cluster nodes. The command sequentially polls the files on the accessible nodes. If the command is rejected by a node when requesting a file, all previously set lock files are released again and the command terminates with an error.

OPTIONS

–C *clos_name*

The name of the cluster is selected in the *clos_name* argument using this option. If this option is not specified, the name from the $CLOSNAME environment variables is used, or if this is not set, the name is taken from the /var/clos/.CLOSdefault file.

–a Enabling A mode (do not abort).

The command is not aborted in this operating mode if one of the selected nodes cannot be accessed. In this case, a warning

WARNING: NODE *tag* not reachable

is issued for the node and sent to standard error output, and no error is indicated in the return value.

A warning is also issued if a tag name was specified for a target node that is not configured in the cluster:

WARNING: NODE *tag* not configured

In A mode, this does not result in the command being aborted, rather it terminates with the warning

WARNING: No selected NODE usable

if none of the selected target nodes can be accessed. The return value in this case is 0.

–t *taglist*

Selecting the target node(s).

The *taglist* argument is a list of tag names for the target nodes on which the command is executed. A list consists of the tag names separated by commas. A white space character (blank, tab, newline character) terminates the list. If this option is not used, all configured nodes are on the list.

If the –T option was used, all nodes are selected *except* for those contained in the list.

The clos_lock command requests all selected nodes in order to create the lock file. If one of the nodes on the list could not be accessed when called or if A mode is not enabled, clos_lock is terminated with error code 4.

–T Inverting the target node list.

The list of target nodes specified with the –t option is inverted. This means that the lock file is created on all nodes that are *not* on the list. If the list is blank (no –t option), the command is executed on all nodes except the one on which clos_lock was called.

–k *text*

Key text transfer.

Key *text* can be transferred if required. This is the text included in the lock file. When releasing the lock file, this text is compared with the text transferred with the clos_unlock(1M) command. Only when the key text matches, is the lock file removed again.

If no key text is specified, clos_lock uses the local node name as key text.

Note:

The *text* argument is a character string. Please note the rules governing the use of quotation marks so that blanks in the text are correctly interpreted by the shell.

Example:

```
"\"This is text\""
```

generates the character string

```
This is text
```

in the file. In contrast,

```
"This is text"
```

is aborted after the first word:

```
This
```

−v Local test (verification).

This option checks whether the file referenced in the argument of the −p option exists as a lock file on the local node.

−w *sec*

Setting a wait time.

If the request for a lock file is rejected because one has already been assigned, you can use this option to set a time *sec* in seconds, which determines how long the test should be repeated. If the file becomes free within this period of time, the lock file is assigned to the requesting user. If the lock file does not become free during this period of time, the command terminates with errors when the set time expired (return value 5).

−p *pathname*

Name of the lock file. The name is either an absolute pathname or a relative pathname, i.e. relative to the /var/clos/*clos_name*/locks directory. If such a relative pathname is in question, the command that is prefixed by the subdirectory views this directory as a lock file.

Example:

```
cod_sync/vdadmin
```

is the name of the lock file for the vdadmin(8) command. The cod_sync(1M) command then terminates immediately with errors every time it is called.

This option must be specified. It determines the name of the lock file from the argument. If the *pathname* argument for this option is an absolute pathname, then this is the name of the lock file. In the case of relative pathnames, this name is appended to the /var/clos/*clos_name*/locks pathname.

−? Usage of this command.

NOTES

The command terminates with a warning to standard error output if it is called in an environment that cannot be assigned to a CLOS. This may happen if no cluster is defined or if the local node is not configured in the specified cluster. The precise information is given in the warning. The command performs no actions in this case and terminates without errors (return value 0).

DIAGNOSTICS

If the clos_lock command discovers an error situation, this is reported to standard error output with error text. The return value is 0 if the lock files could be created on the desired nodes. If an error is reported or one of the lock files has not been acquired, the return value is less than or equal to 7. In this case, a lock file is not

created on any node.

`0` The following messages are possible in this situation:

"`WARNING: Can't assign CLOS name`". The cluster name could not be established. This can only happen if no `CLOS` was set up with the `clos_adm`(1M) command or if the specified *clos_name* argument does not exist.

"`WARNING: local node` *tag* `in CLOS` *clos_name* `not configured`". The node specified on the *taglist* is not configured in the cluster.

`1` "`illegal option`", "`pathname missing`". Incorrect call.

`2` "`local node` *tag* `in CLOS` *clos_name* `not integrated`". The local node is not integrated in the cluster (`NOT_READY` or `NOT_AVAIL` state).

`4` "`NODE` *tag* `not reachable!`". The caller cannot access the desired node.

`5` "`NODE(s) '`*tag*`' lock file busy`". The lock file already exists on one or more of the desired nodes.

`6` "`NODE(s)` *taglist*`: remote command failed!`". The attempt to create a lock file on one of the desired nodes failed.

`7` "`negative timeout value`" or "`timeout value` *value* `too big`". Incorrect call: The value specified with the `-w` option is outside the permitted range.

FILES

`/var/clos/`*clos_name*`/locks`

SEE ALSO

`clos_unlock`(1M), `cod_sync`(1M), `clos`(7).

Fujitsu Siemens Computers Add-on Reference Manual

clos_notify(1M)
Cluster Operating System

NAME

clos_notify - record changes to the node status in the cluster

SYNOPSIS

clos_notify [–C *clos_name*] [–o *outfile*]

clos_notify [–C *clos_name*] [–d *outfile*]

clos_notify [–C *clos_name*] [–e *outfile*]

clos_notify [–C *clos_name*] [–k]

clos_notify –?

DESCRIPTION

The clos_notify command starts a process on the local node. This process reads the /var/clos/*clos_name*/nodes/clos_state file and shows the node status changes for the specified cluster. The process is deemed to be a daemon process if it was started with the –o or –e option and if it writes the status changes to the file specified with the *outfile* argument. If neither of the two options is used, standard output is used as the output medium. The clos_notify daemon terminates with the –k option or otherwise with the process abort signal SIGINT (CTRL-C).

The clos_notify daemon opens the clos_state file which contains all the node status changes logged on the local node up to this point. Every change is entered as an ASCII line in the clos_state file in the format:

DATE@nodename@STATUS

In this case, *DATE* refers to the time elapsed in seconds since 1.1.1970, *nodename* is the name of the node whose status has changed and *STATUS* is the new node status. The node status can be any of the following:

NOT_RUNNING
: The operating system is not booted on the node

NOT_AVAIL
: The node is not accessible

INTEGRATED
: The node is integrated in the cluster

DISABLED
: The node is disabled for the purposes of communication

REMOVED
: The node is removed fully from the cluster

The clos_notify daemon reads the clos_state file line by line, and passes it unchanged to the output medium until the file has been completely read. The daemon then closes the file and shortens it to a length of 0. It then reopens it and converts it to a FIFO [see fattach(3C)] in order to wait for further entries in the FIFO. These latest changes are once again displayed on the output medium. Once the process has ended, the clos_state file again becomes a normal file which is distributed to all nodes in the cluster [see mon_send_rq(3-clos)].

The clos_notify daemon can be used in three operating modes:

Command
: This is where the daemon is called without options and standard output is the output medium.

Daemon
: There is where the daemon is started in the background with the –o option.

Monitor

This is where the daemon was started (perhaps a number of times) with the –e option.

Only one clos_notify process for a cluster can run on a node. It is therefore not possible to start the clos_notify command in the different operating modes listed above at the same time. In addition, a clos_notify command cannot be started a number of times in the *Command* and *Daemon* operating modes. If these rules are violated, the command aborts with a return value of 7.

A clos_notify terminates differently in the individual operating modes:

- In *Command* mode, with the process abort signal SIGINT (CTRL-C),
- In *Daemon* mode with the –k option and
- In *Monitor* mode with the –d option, if this was used to disable the last output file.

OPTIONS

–C *clos_name*
: The name of the cluster is selected in the *clos_name* argument using this option. If this option is not specified, the name from the $CLOSNAME environment variable is used or, if this is not set, the name is taken from the /var/clos/.CLOSdefault file.

–d *outfile*
: (disable) This option signals to a daemon process in the *Monitor* operating mode that it is to close one of its output files. The *outfile* argument represents the pathname of the file that is to be closed. The daemon process closes this file and terminates when the last of its output files has been closed.

–e *outfile*
: (enable) This option starts a daemon process in the *Daemon* operating mode. The *outfile* argument represents the pathname of the file to which the node status changes are added. If a daemon has already been started in this operating mode, it is notified of the additional pathname.

–k
: (kill) Ending the active clos_notify process for the cluster addressed. A process in the *Daemon* operating mode is terminated. The process number is determined from the lock file and the SIGTERM signal is sent to the process. After the daemon has closed the clos_state file, it distributes it to all nodes in the cluster.

–o
: (output) This option starts a daemon process in the *Daemon* operating mode. The *outfile* argument represents the pathname of the file to which the node status changes are added. If the attempt to add entries to the file fails, an abort condition exists and the last status change is written to the clos_state file. The daemon then ends.

  A pathname must be specified.

–?
: Usage of this command.

DIAGNOSTICS

If the clos_notify command comes across an error situation before a daemon process is started, this is reported and no action is performed. The return value is 0 if no error was discovered.

Error messages are output with the following return values to standard error output:

1 "Illegal usage". Incorrect call.

2 "Can't get lock". A lock file with the process number cannot be created.

3 "write to outfile failed". Problem with the write(2) system call.

4 "unexpected signal". An unexpected signal was received.

5 "Can't access lockfile". The lock file with the process number cannot be accessed.

6 "Can't access pathname". The clos_state file or the *outfile* output file cannot be accessed.

7 "Can't create MONITOR". A daemon could not be found or configured in the *Monitor* operating mode. The reason is also given.

8 "No command send to monitor". No command could be sent to a daemon in the *Monitor* operating mode. The reason is also given.

9 "`Can't handle daemon type`". It was attempted to execute a command for a daemon in a different operating mode.

If a `clos_notify` daemon discovers an error, it can no longer report this to standard error output because it no longer has a controlling TTY. This is why the accompanying error message is written to the `/var/saf/`*clos_name*`/notify_err` file. The message text, in this case, is prefixed by the date, time and PID of the daemon, and the daemon then terminates.

FILES

`/var/clos/.CLOSdefault`

`/var/clos/`*clos_name*`/nodes/clos_state`

`/var/saf/`*clos_name*`/notify_err`

SEE ALSO

`fattach`(3C), `mon_send_rq`(3-clos), `clos`(7).

Fujitsu Siemens Computers Add-on Reference Manual

clos_send(1M)
Cluster Operating System

NAME

clos_send - distributing files in the cluster

SYNOPSIS

clos_send [–C *clos_name*] [–T] [–t *taglist*] [–i *icmd*] [–p *pcmd*] *pathname*

clos_send –?

DESCRIPTION

The *pathname* file is transferred from the local node to exactly the same position on the remote node. The file is not transferred atomically, i.e. the file is sent as a datagram to every cluster service daemon [see clserver(1M)] on the remote node which then installs the file. No acknowledgement is issued. Any nodes that were not accessible at this point do not receive the data. Any node integrated subsequently must receive the data in a different way.

OPTIONS

–C *clos_name*
: The name of the cluster is selected in the *clos_name* argument using this option. If this option is not specified, the name from the $CLOSNAME environment variable is used or, if this is not set, the name is taken from the /var/clos/.CLOSdefault file.

–i *icmd*
: Executing an *icmd* initial command.

  The –i option is used to specify the pathname for an *icmd* command that is executed before being transferred. This command can be a script that ensures the consistency of the data while it is being transferred. Thus, a daemon could ensure that the *pathname* file is closed.

  A shell with root authorization, which executes the command, is created on the target nodes.

–p *pcmd*
: Executing a *pcmd* termination command.

  The *pcmd* option is used to specify the pathname for a *pcmd* command that is executed after it is transferred. For example, the daemon could ensure that the *pathhname* file is opened again.

  A shell with root authorization, which executes the command, is created on the target nodes.

–t *taglist*
: Selecting the target node(s).

  The *taglist* argument is a list of tag names for the target nodes to which the *pathname* file is to be transferred. A list consists of the tag names separated by commas. A white space character (blank, tab, newline character) completes the list. If this option is not used, the *pathname* file is transferred to all active nodes. This option can be used several times. Invalid tag names are ignored.

  If the –T option was used, the file is transferred to all nodes *except* for those on the list.

–T
: Inverting the target node list.

  The list of target nodes specified using the –t option is inverted. This means that the file is transferred to all nodes that are *not* on the list. If the list is blank (no –t option), the *pathname* file is transferred to all nodes except to those on which clos_send was called.

–?
: Usage of this command.

DIAGNOSTICS

If the clos_send command comes across an error situation locally before the *icmd* or *pcmd* command name is distributed or before the *pathname* file is transferred, this is reported and no data is transferred. The return

value is 0 if no error was discovered.

Since the command sends the *pathname* file by datagram, no messages are issued if errors occur when transferring the commands or executing them on the remote nodes.

Error messages are sent to standard error output:

1 "`Illegal arguments`". Incorrect call.

2 "`Can't use CLOS service! Closmon not running`". The local node is not integrated in the cluster (`NOT_READY` or `NOT_AVAIL` status).

3 "`Can't access _pmpipe!`" The caller cannot write to the `/etc/saf/`*clos_name*`/_pmpipe` file.

4 "`pathname does not exist`". The file to be transferred could not be found or the full pathname was not specified.

5 "`Command does not exist`". A specified command (`-i` or `-p`) was not found.

FILES

`/var/clos/.CLOSdefault`

SEE ALSO

`closmon`(1M), `clserver`(1M), `clos`(7).

Fujitsu Siemens Computers Add-on Reference Manual

clos_sh(1M)
Cluster Operating System

NAME

clos_sh - synchronously execute a shell in the cluster

SYNOPSIS

clos_sh [−C *clos_name*] −i *id* [−a] [−T] [−t *taglist*] [−p *path*] [−z *sec*] *cmd_string*

clos_sh −?

DESCRIPTION

Using the clos_sh command, a CLOS subsystem [see clos(7)] can execute a shell command [see sh(1) and ksh(1)] on all specified and accessible nodes in the cluster. The command is executed synchronously and in parallel on the nodes. There is no guarantee that the command will be executed within a specified period of time. clos_sh distributes the command via the node channels [see clos(7)] and waits for acknowledgements.

A check is carried out to determine the accessibility of the selected nodes. If a node is selected that is not currently accessible, the command terminates with an error if the −a option is not set. Transmission of the *cmd_string* command only begins when all nodes are accessible or A mode is enabled [see the −a option]. If a command could not be executed on a node or if the attempt to transfer it to another node failed, an appropriate flag is set in the clos_sh return value.

A one-line acknowledgement is sent to standard output for every selected node. If the command was executed successfully, the acknowledgement reads:

NODE *tag*: Command done! exit status *Number*

The *cmd_string* argument represents a shell command string that can be executed under Reliant UNIX. This string is distributed via the Closmon monitor [see closmon(1M)] to the selected nodes where it is executed. In order to execute the command, the callers' default shell is started on every node in their $HOME directories with the appropriate access rights. The $PATH, $HOME and $USER environment variables are taken from the /etc/passwd file on the respective node. The command return value is reported to standard output along with the *number* status.

OPTIONS

−C *clos_name*
: The name of the cluster is selected in the *clos_name* argument using this option. If this option is not specified, the name from the $CLOSNAME environment variable is used or, if this is not set, the name is taken from the /var/clos/.CLOSdefault file.

−a
: Enabling A mode (do not abort).

  The command is not aborted in this operating mode if one of the selected nodes cannot be accessed. In this case, an appropriate message is output for the node and the accompanying flag is set in the return value.

−i *id*
: Command identification.

  An ID *must* be given with every command that is executed with clos_sh. This ID is any character string and helps the caller to determine the output from the executed program. The ID you choose should ideally be identical to the command name; the ID is mandatory input.

−p *path*
: Pathname of the target directory.

  Using this option, the user can specify a target directory for the results of the commands executed on the individual nodes. The *path* argument can be both absolute and relative to the current directory

assigned to the `clos_sh` user. If this option is not used, the current directory is selected. If the user does not have write access for the directory, the command terminates with an error.

`-t` *taglist*
Selecting the target node(s).

The *taglist* argument is a list of tag names for the target nodes on which the command is executed. A list consists of the tag names separated by commas. A white space character (blank, tab, newline character) completes the list. If this option is not used, all configured nodes are on the list.

The command is executed on all nodes on the *taglist* list. If one of the nodes on the list cannot be accessed or if A mode is not enabled when calling the command, `clos_sh` is terminated with an error.

If the `-T` option was used, the command is executed on all nodes *except* for those on the list.

`-T` Inverting the target node list.

The list of target nodes specified using the `-t` option is inverted. This means that the *cmd_string* command is executed on all nodes that are *not* on the list. If the list is blank (no `-t` option), the command is executed on all nodes except the one on which `clos_sh` was called.

`-z` *sec*
Enabling time monitoring (timeout).

Specification of a timeout is optional. This means that the `SIGINT` signal is sent to all processes created by the command a maximum of *sec* seconds after the last command is started on all nodes. The processes normally terminate and the return value for the command is forwarded to the `clos_sh` command. This command then ends when the return value is received by each node. The return value for the `clos_sh` command is 8.

`-?` Usage of this command.

OUTPUT FORMATS

The results of the commands started by `clos_sh` are returned in three files on the local node for every node. The names of the result files have the following format:

For standard output:
path`/`*tag*`.`*id*`.out`

For standard error output:
path`/`*tag*`.`*id*`.err`

For the return value:
path`/`*tag*`.`*id*`.ret`

DIAGNOSTICS

If the `clos_sh` command comes across an error situation locally before the command is distributed, this is reported and the command is not executed on any of the nodes on the *taglist* list. The return value is 0 if no error is discovered. If an error is reported, the return value must be interpreted as follows: Return values are less than or equal to 7. In these cases, no command has been transferred or started. A message is directed to standard error output and the return values have the following meaning:

`1` "`illegal option`", "`Command missing`". Incorrect call.

`2` "`local node not configured`". The local node is not configured in the cluster.

`3` "`local node not integrated`". The local node is not integrated in the cluster (`NOT_READY` or `NOT_AVAIL` status).

`4` "`Poll system call failed!`". The `poll(2)` system call failed.

`5` "`Open channel to NODE` *tag* `fails!`" The caller cannot open the specified node channel.

`6` "`Can't access` *path*". The program user cannot access the specified directory in order to write the output files.

`7` "`Creating output channel to NODE` *tag* `fails`". The program was unable to set up a

connection to the executing command on the remote node.

When transmission and execution of the command has begun, an acknowledgement is expected from every node and is reported to standard output. If an error occurred on one of the node channels when transferring or executing the command, the acknowledgement text has the following format:

`NODE` *tag*`:` *acknowledgement text*`! errno` *number* `->` *error text*

The return value then contains the following values:

`8` "`*** NODE` *TAG*`: command-ID` *id* `interrupted`". Either the timeout or a signal aborted the commands. An appropriate message is sent to standard error output.

`16` "`Can't send cmd request`". The command could not be issued on the node channel.

`32` "`Wrong ID on CLOS_ACKN response`". `CLOS` processed the commands incorrectly.

`64` "`Can't create output channel`". A connection could not be established to an output file on one of the nodes.

`128` "`Command Transmission failed`". Transmission of the command failed on one of the channels.

FILES

`/var/clos/.CLOSdefault`

`/etc/passwd`

SEE ALSO

`ksh`(1), `sh`(1), `closmon`(1M), `poll`(2), `passwd`(4), `environ`(5), `clos`(7).

Fujitsu Siemens Computers Add-on Reference Manual

clos_unlock(1M)
Cluster Operating System

NAME

clos_unlock - synchronously remove a lock file in the cluster

SYNOPSIS

clos_unlock [–C *clos_name*] [–T] [–f | –k *text*] [–t *taglist*] –p *pathname*

clos_unlock –?

DESCRIPTION

Using the clos_unlock command, a CLOS subsystem [see clos(7)] can remove a lock file on all specified nodes in the cluster. This occurs after the lock file key text has been compared with the command key text (–k). If the contents of the lock file and the key text do not coincide, the lock file is not removed and the error message

ERROR: *tag* not unlocked!

is sent to standard error output. The return value is marked with error code 4.

If the user is not authorized to remove the lock file on one of the nodes, the message

ERROR: *tag* access denied! not unlocked

is sent to standard error output and the return value is marked with error code 8.

If the specified lock file is not available on one of the nodes, the command is ignored and no error is displayed.

If none of the selected target nodes can be accessed, the command terminates with the warning

WARNING: No selected node usable

The return value in this case is 0.

A warning is also issued if a tag name is specified for a target node that is not configured in the cluster:

WARNING: NODE *tag* not configured

When removing lock files on the remote nodes, situations can arise that are reported as warnings to standard error output.

If one of the specified nodes cannot be accessed at present, the warning

WARNING: *tag* not reachable!

is displayed and no error code is marked. A warning is also issued if the attempt to issue the synchronous sync_cmd(1M) command fails five times:

WARNING: NODE(s) *taglist*: Try unlock with no response

In these cases, clos_unlock tries to remove the lock file with the help of the Cluster Service [see clserver(1M)] without receiving a response from the relevant node [see clos_cmd(1M)]. The return values from this command are then 0.

If the file did not exist on one of the selected nodes, the warning

WARNING: NODE *tag* -> lockfile doesn't exist!

is output.

OPTIONS

–C *clos_name*

The name of the cluster is selected in the *clos_name* argument using this option. If this option is not specified, the name from the $CLOSNAME environment variable is used or, if this is not set, the name is taken from the /var/clos/.CLOSdefault file.

–t *taglist*

Selecting the target node(s).

The *taglist* argument is a list of tag names for the target nodes on which the lock file is to be removed. A list consists of the tag names separated by commas. A white space character (blank, tab, newline character) completes the list. If this option is not used, all configured nodes are on the list.

If the –T option was used, all nodes are selected *except* for those on the list.

–T Inverting the target node list.

The list of target nodes specified using the –t option is inverted. This means that the lock file is deleted on all nodes that are *not* on the list. If the list is blank (no –t option), the command is executed on all nodes except the one on which clos_unlock was called.

–f Forced unlock.

The contents of the lock file are not compared with the key text in this mode and the file is removed if possible.

–k *text*

Key text transfer.

Key *text* can be transferred if desired. This is the text that is compared with the contents of the lock file. The clos_lock(1M) command inserted this text when the lock file was created. Only when the key text coincides, is the lock file deleted.

If no key text is specified, clos_unlock uses the local node name as key text.

Note:

The *text* argument is a character string. Please note the rules governing the use of quotation marks so that blanks in the text are correctly interpreted by the shell.

Example:

```
"\"This is text\""
```

generates the character string

```
This is text
```

in the file. In contrast,

```
"This is text"
```

is aborted after the first word:

```
This
```

–p *pathname*

Name of the lock file. The name is either an absolute pathname or a relative pathname, i.e. relative to the /var/clos/*clos_name*/locks directory.

This option must be specified. It is used to determine the name of the lock file from the argument. If the argument for this *pathname* option is an absolute pathname, then this is the name of the lock file. In the case of relative pathnames, this name is added to the /var/clos/*clos_name*/locks pathname.

–? Usage of this command.

NOTES

The command terminates with a warning to standard error output if it is called in an environment that cannot be assigned to a CLOS. This may happen if no cluster is defined or if the local node is not configured in the specified cluster. The precise information is given in the warning. The command performs no actions in this case and terminates without errors (return value 0).

DIAGNOSTICS

If the clos_unlock command comes across an error situation, this is reported to standard error output with error text. The return value is 0 without a message if the lock files could be removed on the required nodes.

0 The following messages are possible in this situation:

"WARNING: Can't assign CLOS name". The cluster name could not be established. This can only

happen if no CLOS was set up with the clos_adm(1M) command or if the specified *clos_name* argument does not exist.

"WARNING: local node *tag* in CLOS *clos_name* not configured". The node specified on the *taglist* is not configured in the cluster.

1 "illegal option" and "illegal usage of option -k and -f". Incorrect use of options. The message "illegal usage of option -k and -f" in particular indicates that both options were used simultaneously.

"pathname missing". This indicates that the lock file was not specified as was required.

2 "local node *tag* in CLOS *clos_name* not integrated". The local node is not integrated in the cluster (NOT_READY or NOT_AVAIL status).

The return value contains the following markers if the following situations were identified on one of the nodes:

4 The attempt to compare the key text failed and the lock file was not deleted on one of the nodes.

8 The lock file could not be deleted on one of the nodes because of insufficient access rights, for example.

If both flags can be reported simultaneously, the return value is 12.

FILES

/var/clos/*clos_name*/locks

SEE ALSO

clos_cmd(1M), clos_lock(1M), clserver(1M), sync_cmd(1M), clos(7).

Fujitsu Siemens Computers Add-on Reference Manual

closmon(1M)
Cluster Operating System

NAME

`closmon` - communication monitor for Cluster Operating System

SYNOPSIS

`/usr/lib/saf/closmon` [`–T` *level*]

`/usr/lib/saf/closmon –?`

DESCRIPTION

The `Closmon` monitor is a port monitor based on TTY-STREAMS as provided in System V Release 4 by SAF (Service Access Facility). The `Closmon` monitor is designed in such a way that it normally only runs under the control of the SAC (Service Access Controller) [see `sac(1M)`]. The usual commands for a port monitor can be used with one exception: The format of the `_pmtab` entries is not disclosed and, therefore, the `pmadm -a` command can only be called indirectly using the commands of the `CLOS` Cluster Operating System [see `clos(7)`]. For every *clos_name* cluster, an incarnation of the `Closmon` monitor is started with *clos_name* as the tag name for the port monitor [see `sacadm(1M)`].

The purpose of the `Closmon` monitor is to maintain communication between the nodes of a cluster. This monitor obtains the day and node names from its `/etc/saf/`*clos_name*`/_pmtab` database [see `clos(7)`] and sets up a socket connection to every active node in the cluster. A connection is only set up if a `Closmon` monitor has also been started on the remote node. All nodes in the cluster on which a monitor is started have the status `INTEGRATED` and all monitors are interconnected.

A connection is established in two phases. The monitor creates a socket port [see `socket(3N)`] for every remote node and sends this port number to the Cluster Service daemon [see `clserver(1M)`] of the respective node. This daemon rejects the connection request if no `Closmon` monitor is active on its local node. However, if the `Closmon` monitor is active, it receives a request from the cluster server to switch a connection to the specified port. The initiating monitor waits meanwhile with the `accept(3N)` system call for the connection to be established. If the remote node was not integrated, the monitor does not wait any longer. If a node is subsequently integrated, the active monitor again requests that a connection be set up from its remote nodes by distributing port numbers again. These connection setup requests are now fulfilled. This 2-phase connection setup protocol enables the `Closmon` monitor to determine the status of the remote nodes at any time.

When starting up a monitor on a node, the `Closmon` monitor distributes its status and the node system data to all configured nodes in the cluster using the Cluster Service daemon [see `clserver(1M)`]. Consequently, every node has a `/var/clos/`*clos_name*`/nodes/`*tag_name* directory under which the monitor keeps the `system` files containing system data and the `state` file indicating the status. If the monitor enters the `DISABLED` state or if it terminates (`NOT_AVAIL`), it distributes this new status again to all the nodes in the `state` file.

When starting up a monitor, a file bearing the node tag name is created in the `/dev/clos/`*clos_name* directory for every configured node. There are no access rights for this file initially, assuming a connection has not yet been set up to the respective node. Only when the connection is set up to a node (`ESTABLISHED`) is the file converted to a FIFO and the user receives read and write access. These FIFOs to the nodes are called "node channels". Every application process that issues an `open` system call, receives a private channel for the respective node. If the connection to the remote node breaks down, the user receives an appropriately set signal as is the norm with STREAMS devices. The node channels can be edited using the various control functions [see (3-clos) manpages].

The `Closmon` monitor keeps a `/var/saf/`*clos_name*`/log` log file in which its start, end and all notable (erroneous) events are entered along with the date. The same events are entered under the Log 3.0 component number 1080019 and error number 100 in the logging system logbook.

The monitor can also be operated in trace mode. It is either switched to this status when it is started by selecting the –T option or each of the trace levels is subsequently increased by one if the process receives the SIGUSR1 (16) signal. The trace data is written to the /var/saf/*clos_name*/trace file. The trace is switched off when the SIGUSR2 (17) signal is received. Up to 4 trace levels are possible.

ADMINISTRATION

The Closmon monitor can be administered, on the one hand, with standard commands such as sacadm(1M) and pmadm(1M) . However, it is highly recommended that you use the commands specially provided to administer a cluster and the accompanying CLOS [see clos(7)]. We would like to draw your attention in particular to the node_adm(1M) command for configuring and controlling a cluster.

OPTIONS

–T *level*

Starting the monitor with the trace on.

The trace is already switched on when the monitor is started. Any child processes in the meantime write their own trace files which still add the process ID as the ending to the name. The higher the level, the more output that follows. The various trace levels are set using the *level* argument.

When the SIGUSR1 (16) signal is received, the trace level is increased by 1. The trace is enabled if it was not already switched on. Similarly, the trace can be disabled using the SIGUSR2 (17) signal.

–? Usage of this command.

COMMAND INTERFACE

Using the FIFOs in the /dev/clos/*clos_name* directory, every application can call functions that result in actions on the respective node. These functions are stored in the libclos.a library. In addition, the monitor can use its /etc/saf/*clos_name*/_pmpipe command FIFO to cause various commands to be executed locally (see also **RECOVERY MECHANISM**). The error output associated with these commands is stored in the /var/clos/errmsg/closmon file.

RECOVERY MECHANISM

The Closmon monitor supports a recovery mechanism in relation to actions being executed on the nodes while one or more nodes are *not* INTEGRATED. If these commands are saved in a shell script file and are stored in the /var/clos/*clos_name*/recover/*tag_name* directory under any name, the monitor executes this script when a new connection is established for the first time to the *tag_name* node. The script is then removed from the directory if the script end status is 0. This procedure is used intensively when synchronizing the CLODB Cluster Object Database [see clodb(7)] in the cod_sync(1M) command.

DIAGNOSTICS

If the Closmon monitor comes across an irreparable error situation, it reports this error in the /var/saf/*clos_name*/log file. The monitor then terminates with the following return values and also enters the symbolic names in /var/saf/*clos_name*/log:

1 Incorrect monitor call.

2 Incorrect call: The monitor was started several times.

3 Situation error: The monitor could not edit its /etc/saf/closmon/_pid file.

4 Situation error: The field configured for the monitor for poll descriptors is too small. A recompilation is necessary.

5 Situation error: The monitor was started with an illegal initial status [see sac(1M)].

6 Situation error: The monitor cannot change the number of open file descriptors. The setrlimit(2) system call fails.

7 Situation error: The monitor unexpectedly received a signal.

8 Situation error: The monitor cannot edit the /etc/saf/*clos_name*/_pmpipe pipe on which it accepts commands.

9 Situation error: The monitor is woke from the `poll` system call with incorrect arguments. In this case, a core file is written to `/etc/saf/`*clos_name*.

10 Situation error: During a consistency check, the monitor comes across an internal programming error and then terminates. In this case, a core file is written to `/etc/saf/`*clos_name*.

11 Situation error: The monitor cannot edit the ports reserved for the Cluster Operating System.

12 Situation error: The monitor has received a message that it is unable to interpret.

FILES

`/dev/clos/`*clos_name*

`/etc/saf/`*clos_name*`/_pid`

`/etc/saf/`*clos_name*`/_pmpipe`

`/etc/saf/`*clos_name*`/_pmtab`

`/etc/saf/`*clos_name*`/_sacpipe`

`/var/saf/`*clos_name*`/log`

`/var/saf/`*clos_name*`/trace`

`/var/clos/errmsg/closmon`

`/var/clos/`*clos_name*`/recover/`*tag_name*

`/var/clos/`*clos_name*`/nodes`

`/var/clos/`*clos_name*`/nodes/`*tag_name*`/state`

`/var/clos/`*clos_name*`/nodes/`*tag_name*`/system`

SEE ALSO

`clserver`(1M), `cod_sync`(1M), `node_adm`(1M), `pmadm`(1M), `sac`(1M), `sacadm`(1M), `open`(2), `poll`(2), `setrlimit`(2), `accept`(3N), `socket`(3N), `clodb`(7), `clos`(7).

"Programmer's Guide: Networking Interfaces, chap. 13".

"Programmer's Guide: STREAMS".

Fujitsu Siemens Computers Add-on Reference Manual

closname(1M)
Cluster Operating System

NAME

`closname -` converting tag names in the cluster

SYNOPSIS

`closname` [`–C` *clos_name*] [`–v`] `host` *hostname*

`closname` [`–C` *clos_name*] [`–v`] `tag` *tagname*

`closname –?`

DESCRIPTION

The `closname` command is used to convert node names.

By assigning the `host` argument to `closname`, the command receives a name as an argument which it interprets as a network name for a node. `closname` searches the local `/etc/saf/`*clos_name*`/_pmtab` file for the node tag name associated with this network name. Permitted network names can be the official hostname, the alias name or the Internet address of a node. The `localhost` or `127.0.0.1` names are also permitted for the local nodes.

The name is interpreted as a tag name when the `tag` argument is set, and the accompanying network name is determined from the `/etc/saf/`*clos_name*`/_pmtab` file.

The command sends the name it has identified to standard output.

OPTIONS

`–C` *clos_name*
: With this option, the name of the cluster is selected in the *clos_name* argument. If this option fails, the name from the `$CLOSNAME` environment variable is used or, if this is not set, the name is taken from the `/var/clos/.CLOSdefault` file.

`–v` Long output format

With this option, the line

`TAG:` *tagname* `HOST:` *hostname*

is sent to standard output instead of the name determined.

`–?` Usage of this command.

DIAGNOSTICS

The `closname` command terminates with the return value 0 if no errors occurred. Error messages are sent to standard error output along with the following return values:

`1` "`Illegal arguments`". Incorrect call.

`2` "`HOST` *hostname* `not found`". The specified network name was not found.

`3` "`TAG` *tagname* `in CLOS '`*clos_name*`' not found`". The specified tag name was not found.

`4` "`Illegal parameter` *parm*". Neither `tag` nor `host` was used as the keyword, or the *clos_name* cluster was not found.

FILES

`/etc/saf/`*clos_name*`/_pmtab`

Fujitsu Siemens Computers Add-on Reference Manual

clserver(1M)
Cluster Operating System

NAME

clserver - cluster service daemon

SYNOPSIS

/usr/lib/saf/clserver [–T *level*]

/usr/lib/saf/clserver –?

DESCRIPTION

The clserver cluster service daemon is a utility that is always started by the Internet daemon [see inetd.conf(4)] when a datagram [see udp(7)] is received on the cluster port (8890/udp) that is unique throughout the system. The various commands of the CLOS Cluster Operating System [see clos(7)] and the Closmon monitor [see closmon(1M)] send queries to the cluster service daemon which, in turn, responds on the port specified in the query. The cluster service daemon processes the following services:

RQ_CONNECT
: This query indicates that a Closmon monitor on another node wishes to set up a connection to the local node. The cluster service daemon asks the local Closmon monitor for the desired cluster. If this monitor is not started, this situation is reported back to the requester.

RQ_FILE
: A requester on another node wishes to send a file to the local node. The CLOS file transfer protocol is used until the file is transferred.

RQ_SHELLCMD
: A requester wants the daemon to start a shell and transfer a character string as a command. For this purpose, the name of the requesting node is set in the $REQUESTERTAG environment variable and the name of the cluster is set in the $CLOSNAME environment variable. Standard error output is redirected by the daemon to the /var/clos/errmsg/clos_service file.

RQ_STATE
: The requesting monitor informs the local node of its status. This is entered by the daemon in the /var/clos/*clos_name*/nodes/*tag_name*/state file and, if it has changed, an appropriate entry including the date is made in the /var/clos/*clos_name*/nodes/clos_state file. The daemon informs the requester of the date of its COD archive and its modify file under /var/clos/*clos_name* [see clos(7)].

RQ_EXEC
: A requester wants the daemon to execute a character string as a command. For this purpose, the tag name of the requesting node is entered in the $REQUESTERTAG environment variable and the cluster name is set in the $CLOSNAME environment variable. Standard error output is redirected by the daemon to the /var/clos/errmsg/clos_service file.

RQ_UNAME
: The requester wants to determine the node system name. The daemon executes the uname(2) system call and enters the utsname structure received in the response. It also indicates in the response whether or not this node name has already been used locally in the cluster of the requesting monitor.

RQ_SV_STATE
: The daemon informs the requester of the date of its COD archive and its modify file under /var/clos/*clos_name* [see clos(7)].

ADMINISTRATION

To administer the cluster service daemon, the system-wide cluster port must be configured using default commands by making an entry in the /etc/services file. The start of the cluster service daemon must then

be entered along with its arguments in the /etc/inetd.conf Internet daemon configuration file, and the inetd(1M) daemon must be requested with the SIGHUP signal [see signal(5)] to read its database again. These settings are entered when installing the Cluster Operating System basic package. Only when the 8890 port number is used by other components, does the administrator have to decide on another number which must nevertheless be the same on all nodes in the cluster.

OPTIONS

-T Starting the daemon in trace mode.

When the daemon is started, the trace is switched on. Any child process in the meantime write to the same /var/saf/clos_service/trace trace file.

-? Usage of this command.

DIAGNOSTICS

If the clserver cluster service daemon comes across an irreparable error situation, it reports this situation in the /var/saf/clos_service/log file. It then terminates with the following return values and also enters the symbolic names in /var/saf/clos_service/log:

1 Incorrect daemon call.

2 Incorrect call: The daemon received an incorrect request.

4 Situation error: The daemon could not transfer the file or execute the command requested because it could not access a pathname.

9 Situation error: The daemon could not read the request or send the caller a response because there were network problems.

FILES

/etc/inetd.conf
/etc/saf/inetd/core
/etc/services
/var/clos/*clos_name*/COD
/var/clos/*clos_name*/modify
/var/clos/*clos_name*/nodes/clos_state
/var/clos/*clos_name*/nodes/*tag_name*/state
/var/clos/errmsg/clos_service
/var/saf/clos_service/log
/var/saf/clos_service/trace

SEE ALSO

closmon(1M), inetd(1M), uname(2), signal(5), clos(7), udp(7).

Fujitsu Siemens Computers Add-on Reference Manual

cluster_rebuild(1M)

NAME

`cluster_rebuild -` command to manually restore an SPS cluster

SYNOPSIS

`cluster_rebuild` [*option ...*]

DESCRIPTION

There are two potential scenarios to consider in the event that the "Node Alive Timer" in the ICF layer of a node expires, indicating that another *X* node may no longer be alive:

- If RMS is not installed on the nodes in the SPS cluster, the *X* node is reset by SPS software on the basis of the EIP. The Event Notification Subsystem (ENS) subsequently produces "Node Down Events" on the remaining nodes.
- If RMS is installed on the nodes in the SPS cluster, RMS is informed under normal conditions that the *X* node is no longer responding. RMS then resets this *X* node and requests that the Event Notification Subsystem (ENS) send "Node Down Events" for the remaining nodes.

However, if in this situation where an *X* node is no longer responding RMS is now installed on the other nodes but is not started or has just been stopped, the entire ENS subsystem is blocked since it is waiting for RMS to resolve the situation. A system administrator can intervene manually in this case and resolve the situation by starting the

`cluster_rebuild –k`

command on the node for which the

`RMS not running dead_nodes=<..> send no sig %d to ...`

message appeared on the console.

The *X* node is then reset and "Node Down Events" are issued for the remaining nodes.

OPTIONS

`–k` Resets the *X* node that is no longer responding and resolves the situation (sends "Node Down Events" for the remaining nodes).

`–n` Resolves the situation (sends "Node Down Events" for the remaining nodes), without resetting the *X* node.

Warning:

The `–k` option should always be used under normal conditions to ensure that the *X*node is reset before the "Node Down Events" are sent.

The `–n` option should really only be used if the *X* node was already reset and is now rebooting.

Fujitsu Siemens Computers Add-on Reference Manual

cod_check(1M)
Cluster Operating System

NAME

cod_check - checking the CLODB in the cluster

SYNOPSIS

cod_check [–C *clos_name*] [–a] [–T] [–t *taglist*] [–cm] [–l]

cod_check [–C *clos_name*] [–a] [–T] [–t *taglist*] –e

cod_check [–C *clos_name*] [–a] [–T] [–t *taglist*] [–cm] –y

cod_check –?

DESCRIPTION

This command is used to check and compare the archived and installed CLODB Cluster Object Databases [see clodb(7)] on all selected nodes in the cluster. The system administrator can arrange to have the date of the COD archive and/or the date of the last change to the installed CLODB output from each node. In addition, the system administrator can perform a consistency check on each node. All commands indicate whether or not the requested results are the same on all selected nodes. If the result is only different on one node, 1 is the value returned. Errors are displayed with return values greater than 1 and no check is performed.

OPTIONS

–C *clos_name*

The name of the cluster is selected in the *clos_name* argument using this option. If this option is not specified, the name from the $CLOSNAME environment variable is used or, if this is not set, the name is taken from the /var/clos/.CLOSdefault file.

–a Enabling A mode (do not abort).

The command is not aborted in this mode if one of the selected nodes cannot be accessed. In this case, a warning

WARNING: NODE *tag* not reachable

is issued for the node and sent to standard error output and no inconsistency is signaled in the return value.

A warning is also output if a tag name was specified for the target nodes that is not configured in the cluster:

WARNING: NODE *tag* not configured

This does not result in an abort situation in A mode. However, the command terminates with the warning

WARNING: No selected NODE usable

if none of the selected target nodes can be accessed. No inconsistency is therefore signaled in the last two cases.

–c Choosing the COD archive.

Only the archives of the selected nodes are chosen with this option. This choice is only valid when listing the date (–l option) and when determining the most recent archive (–y option). This option is ignored with the remaining commands.

–e Performing a consistency check.

Using this command, a consistency check is performed on every selected node by calling codadm(1M) with the –e option where the result is displayed in the following form:

tagname:*result* (*retval*)

In this case, *tagname* is the node name and *result* is either the character string "`Installed CLODB and COD are consistent!`" if evaluation on the node did not identify any discrepancy between the installed `CLODB` and the `COD` archive or "`Installed CLODB and COD are inconsistent`" if a difference was detected. The *retval* value in brackets is the return value generated by the `codadm` command. If the *retval* value is the same for all selected nodes, the value 0 is returned. If an inconsistency is identified on a node, the value 1 is returned and an appropriate message is sent to standard error output.

`-l` Listing the dates.

This command is used to list the dates of the objects selected with the `-c` and `-m` options for all selected nodes. If no object was selected using these options, all dates are listed and displayed in the following format:

tagname`:`*object sec* `==>`*date*

In this case, *tagname* is the node name, *object* is either the `COD` or `modify` archive for the installed database, *sec* is the date in seconds since 1.1.1970 and *date* is the `date(1)` format. If all objects have the same date, the value 0 is returned. If one or more dates differ, the value 1 is returned and an appropriate message is sent to standard error output.

`-m` Choosing the modification date for the installed `CLODB`.

With this option, you can choose the date of update for the installed `CLODB` for the selected nodes. This selection is only valid when listing the date (`-l` option) and when establishing the most recent date (`-y` option). This option is ignored with all other commands.

`-t` *taglist*

Selecting the target node(s).

The *taglist* argument is a list of tag names for the target nodes on which the command is executed. A list consists of the tag names separated by commas. A white space character (blank, tab, newline character) completes the list. If this option is not used, all configured nodes are on the list.

`-T` Inverting the target node list.

The list of target nodes specified using the `-t` option is inverted. This means that all nodes that are *not* on the list are selected. If the list is blank (no `-t` option), all nodes are selected except for the one on which the `cod_check` command was called.

`-y` Determining the most recent date.

Using this command, the dates of the objects selected with the `-c` and `-m` options are compared for all selected nodes and the most recent date is identified. If no object was selected with these options, the dates of all objects are checked and the most recent date is displayed in the following format:

tagname`:`*object sec* `==>`*date*

In this case, *tagname* is the node name, *object* is either the `COD` or `modify` archive for the installed database, *sec* is the date in seconds since 1.1.1970 and *date* is the `date(1)` format. If all examined objects have the same date, the value 0 is returned. If one or more dates differ, the value 1 is returned and an appropriate message is sent to standard error output.

`-?` Usage of this command.

DIAGNOSTICS

The `cod_check` command terminates with the return value 0 if no errors have occurred. In an inconsistency was detected among the selected objects, the return value 1 is output without any additional message. Any other error messages are sent to standard error output.

`1` "`COD/modify dates are inconsistent`". The results list shows that at least one date is different from those of the other nodes. The (`COD` and/or `modify`) objects to be checked are derived from the selected options.

`2` "`illegal usage->`". An error was recognized in the command syntax. The arrow is followed by a reference to the error.

3 "`Can't assign CLOS name`". The name of the cluster cannot be determined. As it happened, the `/var/clos/.CLOSdefault` file was missing.

4 "`NODE` *tagname* `not reachable`". The selected node cannot be accessed for the evaluation command.

5 "`NODE` *tagname* `not configured`". The selected node is not accessible for the evaluation command since it is not contained in the selected cluster.

6 "`This NODE is not reachable`". The local node cannot be accessed for the evaluation command.

7 "`you need to be a superuser for COD evaluation!`". A `CLODB` can only be evaluated by the superuser.

FILES

`/var/clos/.CLOSdefault`

`/var/clos/`*clos_name*`/modify`

`/var/clos/`*clos_name*`/COD`

SEE ALSO

`codadm`(1M), `clodb`(7).

Fujitsu Siemens Computers Add-on Reference Manual

cod_prototype(1M)
Cluster Operating System

NAME

cod_prototype - evaluation of prototypes of the cluster object database

SYNOPSIS

cod_prototype –a *clos_name*

cod_prototype –i *clos_name*

cod_prototype –r *clos_name*

cod_prototype –u *arg clos_name*

cod_prototype –?

DESCRIPTION

This command is used to evaluate the COD_prototype(4) files for a particular situation, in order to configure, remove or update a cluster object database CLODB [see clodb(7)]. Only those actions whose field entry *CLOS_name* corresponds to the *clos_name* command parameter are considered for the respective COD_prototype file entries.

cod_prototype should be invoked in order to install an application which wants to store data in CLODB. If no cluster is configured, cod_prototype will fail, though this should not cause the installation to abort. If the cluster is configured at a later stage, the clos_adm(1M) command provided for this purpose invokes cod_prototype and the actions planned for CLODB are then executed.

If a node is added to a cluster with the node_add(1M) command, cod_prototype is invoked with the –a option on each newly added node. All COD_prototype files with an a flag entry become effective.

If a node is removed from a cluster using the node_rm(1M) command, cod_prototype is invoked with the –r option on the node to be removed. cod_prototype is invoked with the –u option on the remaining nodes.

The situations envisaged by the Cluster Operating System CLOS [see clos(7)], where the cod_prototype command is invoked, are listed below. In addition, an application may also recognize other situations where it may make sense to invoke the command.

Warning: The command only impacts the nodes locally.

OPTIONS

–a When a new node is added to the cluster ("add node"), all entries with a a flag are executed as commands.

–i Install CLODB.

This option is used to install the CLODB of the cluster with the name specified in the *clos_name* parameter. This means that all COD_prototype files are then evaluated to determine whether they describe actions for the selected cluster.

Actions that correspond to the flags in the COD_prototype files could be:

c cod_prototype copies the file referenced in the *file* field to CLODB. The absolute pathname of the file is appended to the CLODB home directory /var/clos/*clos_name*/root. The rights for the new file are taken from the *args* field of the COD_prototype file entry. The file can only be copied if the destination file does not yet exist.

i cod_prototype executes the file referenced in the *file* field as a command with the arguments in the *args* field of the COD_prototype file entry.

l cod_prototype creates a symbolic link from the file referenced in the *file* field to the CLODB home directory /var/clos/*clos_name*/root. The pathname of the new file is derived from the name of the home directory and the pathname of *file*. The link is only set if the destination file

does not yet exist.

p cod_prototype creates a symbolic link from the file referenced in the *file* field to the CLODB home directory /var/clos/*clos_name*/root. The pathname of the new file is derived from the name of the home directory and the pathname of *file*. The link is only set if the destination file does not yet exist.

−r Remove CLODB.

This option removes the CLODB of the cluster with the name specified in the *clos_name* parameter. This means that all COD_prototype files are then evaluated to determine whether they describe actions for the selected cluster for this purpose. Permanent symbolic links are preserved. Actions that correspond to the flags in these files include:

c cod_prototype deletes the file referenced in the *file* field in CLODB with the home directory /var/clos/*clos_name*/root.

r cod_prototype executes the file referenced in the *file* field as a command with the arguments in the *args* field of the COD_prototype file entry.

l cod_prototype deletes the file referenced in the *file* field in CLODB with the home directory /var/clos/*clos_name*/root.

−u Update CLODB.

This option is used to update the CLODB of the cluster with the name specified in the *clos_name* parameter. This means that all COD_prototype files are then evaluated to determine whether they describe actions for the selected cluster for this purpose. The *arg* argument is used consistently for %A.

Actions that correspond to the flags in the COD_prototype files include:

u cod_prototype executes the file referenced in the *file* field as a command with the arguments in the *args* field of the COD_prototype file entry.

−? Usage of this command.

DIAGNOSTICS

The cod_prototype command terminates with the return value 0 if no errors occur. Error messages are sent to standard error output.

1 "missing CLOSNAME" or "illegal option". A syntax error was discovered in the command line.

2 "no such CLOS" or "missing CLOSNAME". Either no cluster was specified as a parameter or the cluster specified is not yet configured.

3 "not super user". This command may only be executed by the superuser.

FILES

/var/clos/*clos_name*/root

SEE ALSO

clos_adm(1M), node_rm(1M), clodb(7), clos(7).

Fujitsu Siemens Computers Add-on Reference Manual

cod_send(1M)
Cluster Operating System

NAME

`cod_send` - distributing Cluster Object Database files

SYNOPSIS

`cod_send` [`-C` *clos_name*] `-i` *id* [`-k` *text*] *file_list*

`cod_send` [`-C` *clos_name*] `-r` *tag_name* `-i` *id*

`cod_send -?`

DESCRIPTION

This command distributes files from the local Cluster Object Database `CLODB` [see `clodb(7)`] to all nodes in the cluster. The *file_list* argument in this case represents a list of pathnames relative to the `/var/clos/`*clos_name* directory that are separated by blanks. The *id* argument of the `-i` option represents any character string that uniquely identifies the list of files in *file_list*.

The command requires that a lock file called `cod_sync/`*id* be created already [see `clos_lock(1M)`]. This prevents a `cod_sync(1M)` command being issued at this time in the cluster. If there is no lock file, `cod_send(1M)` terminates with errors.

A recovery script is produced for all unreachable nodes and stored in the `/var/clos/`*clos_name*`/recovery/`*tag_name* directory. When the node is reintegrated, the recovery script is executed by the `closmon(1M)` command and then invokes `cod_send` itself with the `-r` option.

If the lock file exists, the files in *file_list* are copied to all nodes.

OPTIONS

`-C` *clos_name*
: This option is used to select the name of the cluster in the *clos_name* argument. If this option is not specified, the name is taken from the `$CLOSNAME` environment variable, or if this variable is not set, from the `/var/clos/.CLOSdefault` file.

`-i` *id*
: ID for the specified list of files *file_list*.

`-k` *text*
: Key text transfer (key).

  Key *text* can be transferred if required. This is the text included in the lock file. This text is compared with the contents of the lock file before `cod_send` is executed. The command is only executed if the key text matches.

  If no key text is specified, the local node name is used as the key text and compared.

  Note:

  The *text* argument is a character string. Note the quota allocation rules so that blanks in the text are interpreted correctly by the shell [cf. `clos_lock(1M)`].

`-r` *tag_name*
: Recovery mode. This option informs the `cod_send` command that a recovery mechanism is to be performed for the *tag_name* node. In contrast to the normal (no `-r`) `cod_send` command, the files are only copied to the *tag_name* node, if a recovery must actually be performed.

`-?` Usage of this command.

NOTES

The command terminates with a warning to standard error output if it is called in an environment that cannot be assigned to a `CLOS`. This may happen if no cluster is defined or if the local node is not configured in the

specified cluster. The precise information is given in the warning. The command performs no actions in this case and terminates without errors (return value 0).

DIAGNOSTICS

If the `cod_send` command comes across an error situation locally before the files are transferred, this is reported and no data is sent.

The command terminates with the return value 0 if no errors have occurred. Error messages are sent to standard error output.

1 "`illegal options`". Illegal options were used for the call.

2 "`file name missing!`". No pathname was specified for the files.

3 "`timeout for cod_sync recovery`". If `cod_send` is called while a recovery script is active for the `cod_sync(1M)` command, the command waits for `cod_sync` to end. A timeout is initiated for this wait time. If the recovery has not ended before this timeout expires, the command returns this error.

4 "`local NODE not integrated! CLOSMON not running`". The local node is not currently integrated in the cluster.

5 "`No lock set or key doesn't match!`". The lock file for synchronization with the `cod_sync(1M)` command is not set or different text has been entered in the lock file. The `cod_send` command has been called without a lock file or by another user of the same set of files (*file_list*). Exception: In the case of the recovery mechanism, this situation causes an abort without errors.

6 "`you need to be superuser for COD sync!`". A `CLODB` database may only be synchronized by the superuser.

7 "`Can't create file archive!`". A new archive could not be created on the local node. [`tar(1)` provides information on possible causes.]

8 "`Can't send archive!`". The archive created locally could not be distributed to all nodes [for reasons, see `sync_send(1M)`].

9 "`Setup of archive failed 5 times!`". The files could not be copied to the nodes.

10 "`NODE` *tag* `not available for recovery!`". The node, for which the recovery mechanism is to be performed, is currently not integrated.

FILES

`/var/clos/`*clos_name*`/root`

`/var/clos/`*clos_name*`/recovery/`*tag_name*

`/var/clos/.CLOSdefault`

SEE ALSO

`closmon(1M)`, `clos_lock(1M)`, `cod_sync(1M)`, `sync_cmd(1M)`, `sync_send(1M)`.

Fujitsu Siemens Computers Add-on Reference Manual

cod_sync(1M)
Cluster Operating System

NAME

cod_sync - distributing the Cluster Object Database

SYNOPSIS

cod_sync [–C *clos_name*] [–I] [–p]

cod_sync [–C *clos_name*] –r [–I] *tag_name*

cod_sync –?

DESCRIPTION

Using this command, the local CLODB Cluster Object Database [see clodb(7)] is distributed to and installed under the /var/clos/*clos_name*/root directory on all nodes in the cluster.

Using clos_lock(1M) , the command first requests a lock file on all selected and accessible nodes. If the command does not receive this lock file, it terminates with an error without any further actions. A recovery script is generated for all inaccessible nodes and is stored in the /var/clos/*clos_name*/recovery/*tag_name* directory.

If the request for a lock file is successful, a COD archive is created and is then distributed to all accessible nodes where it is installed. If an archive already exists on the local node and the –p option is used, a partial COD.part archive is created, distributed and installed. A new COD archive that is installed consistently is always produced locally.

The effect on the remote node of creating a partial archive (–p) is that the CLODB there is not consistent in all cases.

OPTIONS

–C *clos_name*
: Using this option, the name of the cluster is selected in the *clos_name* argument. If this option is not specified, the name from the $CLOSNAME environment variable is used or, if this is not set, the name is taken from the /var/clos/.CLOSdefault file.

–I
: Installation of the COD archive is suppressed.

  Using this option, the archive created on the initial node is distributed and not installed. Consequently, the CLODB is only consistent on the initial node.

–p
: Creating a partial archive.

  If a COD archive already exists on the local node, a partial archive is created and distributed to all nodes. This results in all local changes on the remote nodes being overwritten with the data to be distributed. Consequently, the CLODB on the remote nodes is not consistent in all cases because the local changes made there are not overwritten. Only the data from the partial archive is distributed and is the same on all nodes.

–r *tag_name*
: Recovery mode. This option informs the cod_sync command that a recovery mechanism is to be carried out for the *tag_name* node. In contrast with the normal cod_sync command, a lock file [see clos_lock(1M)] is only requested in this case on the local and the addressed node. If this request is rejected, the command terminates successfully (return value 0).

  Furthermore, no new archive is created locally. Instead, the archive with the same date as that containing the global $RECOVERPARAM environment variable is used. If the current date of the COD archive is different from the date of the variable, the command is terminated with the return value 0. Otherwise, the archive is transferred to the specified node where it is installed.

–?
: Usage of this command.

NOTES

The command terminates with a warning to standard error output if it is called in an environment that cannot be assigned to a CLOS. This may happen if no cluster is defined or if the local node is not configured in the specified cluster. The precise information is given in the warning. The command performs no actions in this case and terminates without errors (return value 0).

DIAGNOSTICS

If the cod_sync command comes across a local error situation before the (partial) archive is transferred, this is reported and no data is transferred.

The command terminates with the return value 0 if no errors have occurred. Error messages are sent to standard error output.

1 "illegal options". Illegal options were used for the call.

3 "No COD archive available". The archive does not exist, i.e. the /var/clos/*clos_name*/root directory is empty.

4 "local NODE not integrated! CLOSMON not running". The local node is not currently integrated in the cluster.

5 "Lock file busy. Locking failed!". The lock file for synchronizing the CLODB database is already set. The cod_sync command is already active. Exception: This results in an abort without an error in the case of the recovery mechanism.

6 "you need to be a superuser for COD sync!". A CLODB database can only be synchronized by the superuser.

7 "Can't create COD archive!". No new archive could be created on the local node [for reasons, see codadm(1M)].

8 "Can't send archive!". The archive created locally could not be distributed to all nodes [for reasons, see sync_send(1M)].

9 "Installation of COD archive failed!". It was not possible to install the COD archive.

10 "NODE *tag* not available for recovery!". The node, for which the recovery mechanism is to be performed, is currently not integrated.

11 "locking failed for recovery!". The lock file could not be set for the node recovery mechanism.

FILES

/var/clos/*clos_name*/root

/var/clos/*clos_name*/COD

/var/clos/*clos_name*/COD.part

/var/clos/*clos_name*/recovery/*tag_name*

/var/clos/.CLOSdefault

SEE ALSO

closmon(1M), clos_lock(1M), clserver(1M), codadm(1M), sync_cmd(1M), sync_send(1M).

Fujitsu Siemens Computers Add-on Reference Manual

codadm(1M)
Cluster Operating System

NAME

codadm - managing the Cluster Object Database

SYNOPSIS

codadm [–C *clos_name*] –c [–p]

codadm [–C *clos_name*] –i [–p]

codadm [–C *clos_name*] –m

codadm [–C *clos_name*] –e

codadm [–C *clos_name*] –s [*archive*, ...] [–p]

codadm –?

DESCRIPTION

This command enables the Cluster Object Database CLODB [see clodb(7)] to be managed and maintained locally under /var/clos/*clos_name*/root. codadm is used to create, update and install a COD archive and to check the installed database [see clodb(7)].

OPTIONS

–C *clos_name*

The name of the cluster is selected in the *clos_name* argument using this option. If this option is not specified, the name from the $CLOSNAME environment variable is used, or if this is not set, the name in the /var/clos/.CLOSdefault file is used.

–c Creating an archive (create).

All files located under /var/clos/*clos_name*/root are written to the archive. Symbolic links must also be identified and their sources recorded in the COD archive. The date of the /var/clos/*clos_name*/modify file is used to mark the COD archive. This file is also included in the archive, and the COD archive created receives its date so that both dates agree when checked. Furthermore, a new /var/clos/*clos_name*/modify file is created and this is written to the COD archive together with the remaining files and file trees. The date of the /var/clos/*clos_name*/modify file is identical to that of the COD archive.

If a COD archive already exists on a node and the –p option is used, this archive is not overwritten. Instead, only a partial COD archive is created comprising all the changes made with respect to CLODB. The name of this partial archive is always COD.part. When the codadm –c command is called, a CLODB is only deemed to be consistent locally if a complete archive was created.

A lock file is created in the /var/clos/*clos_name*/locks directory to prevent a COD archive from being created several times simultaneously. If you now call another command that results in a COD archive being created, processing is terminated (return value 11).

–i Installing an archive (install).

The archive is installed as a file tree on the node. In this case, the /var/clos/*clos_name*/modify file receives the same date as the COD archive, and CLODB is consistent on the local nodes. The tree that exists in the COD archive is the one that is installed. Files that were changed locally before this installation are overwritten and files that were created in CLODB but not archived disappear. The resulting CLODB is consistent.

If only a partial archive is installed (–p option), a new COD archive is first created from the existing and the partial archive. The partial archive is then reinstalled in CLODB. This means that a previous inconsistency may be retained if the partial archive does not contain any files changed locally. If files are removed in the partial archive, they are not yet removed following a partial installation. Consistency

of CLODB can only be guaranteed after a complete archive has been installed.

In order to prevent codadm –i from being called several times simultaneously, a lock file is created in the /var/clos/*clos_name*/locks directory during installation. If this command is called subsequently, processing is terminated (return value 10).

A lock file is created to prevent a COD archive from being created several times simultaneously. If you now call another command that results in a COD archive being created, processing is terminated (return value 11).

In order to allow the subsystems of the Cluster Operating Systems to carry out a number of measures before CLODB is installed, all executable command files located under /var/clos/*clos_name*/preinstall are executed. Similarly, all command files under /var/clos/*clos_name*/postinstall are executed when the archive has been installed. The name of a file in which all installed files are listed line by line is specified as an argument for each of these command scripts. Each subsystem can therefore determine which measures are to be adopted before and after installation respectively.

–m Marking a CLODB update (modify).

When a CLODB has been updated, the -m option is used to create a new list of files belonging to CLODB along with their checksums in the /var/clos/*clos_name*/modify file. Only real files and symbolic links feature on this list. Directories in the /var/clos/*clos_name*/modify file are ignored. The /var/clos/*clos_name*/modify file receives the current date.

–e Checking the consistency of the local CLODB (evaluate).

This command extracts the /var/clos/*clos_name*/modify file from the COD archive. The first task it performs is to compare the date of the modify file with that of the COD archive. Any discrepancy between dates is reported as an error. In this event, a new checksum file is created and is compared with that from the COD archive. All differences between files are sent to standard output. Any differences between files or dates are reported as errors on standard error output. After the command has been executed, no new files are created locally.

–s Listing the contents of an archive (show).

If no further argument is passed on, the contents of the files are listed in the COD archive. When the –p option is specified, the file contents are listed in the COD.part archive. If an *archive* ... list is specified, the contents of the addressed archive(s) are displayed via standard output.

–p Selecting a partial archive (partial).

When using the -p option, the create (–c), install (–i) and show (–s) functions refer to the archive called COD.part.

–? Usage of this command.

DIAGNOSTICS

The codadm command terminates with the return value 0 provided that no errors have occurred. Error messages are directed to standard error output.

1 "Only one option may be selected" or "illegal option". Too many options were used when calling the command or the call syntax was incorrect.

2 "Can't create archive". Neither the COD nor the COD.part file could be created.

3 "Can't read table of contents in archive", "Can't extract archive" or "Can't read archive". The existing COD or COD.part archive could not be accessed without errors.

4 "Can't create new archive" or "No modify file available". Neither the COD nor the COD.part archive could be created, or the modify file does not exist.

5 "No COD archive available". The archive does not exist.

6 "Can't install partial COD". The COD.part archive could not be accessed without errors.

7 "Can't install previous COD". The old archive could not be installed when a partial installation

was being performed.

`8` "`Can't find archive to install`" or "`No archive available`". The archive to be installed or evaluated could not be found.

`10` "`CLODB is locked for installing`". The database to be installed is already locked for a (competing) installation process.

`11` "`CLODB is locked for COD create`". The database to be created is already locked for a (competing) create process.

`15` "`Archive does not exist`". The archive to be listed does not exist.

The following messages are also output when the evaluation command (`codadm -e`) is called:

`0` "`Installed CLODB and COD are consistent!`". The installed database and archive are consistent on the local node.

`!0` "`Installed CLODB and COD are inconsistent (`*retval*`)`". The installed database and archive are inconsistent on the local node where *retval* is the sum of the following values:

`16` "`Date of COD archive and installed CLODB differ (16)`". The installed database does not match the archive.

`64` "`COD archive with wrong date => inconsistent (64)`". The date of the installed database does not match that of the archive.

`128` "`Following files are removed from CLODB:` *file-list*". When performing an evaluation, it was discovered that files in the installed database were deleted.

"`Following files are new in CLODB:` *file-list*". When performing an evaluation, it was discovered that files had been added to the installed database.

"`Following files in CLODB differ from COD archive:` *file-list*". When performing an evaluation, it was discovered that there were discrepancies between files in the installed database and files in the archive.

The *file-list* output with the error messages is sent to standard output whereas the error message text is directed to standard error output. This enables several messages to arrive simultaneously. The return value is then the sum of the values specified above.

FILES

`/var/clos/`*clos_name*`/root`

`/var/clos/`*clos_name*`/COD`

`/var/clos/`*clos_name*`/COD.part`

`/var/clos/`*clos_name*`/modify`

SEE ALSO

`clodb`(7), `clos`(7).

Fujitsu Siemens Computers Add-on Reference Manual

collector(1M)
Cluster Operating System

NAME

collector - entering event messages in the cluster

SYNOPSIS

collector [–C *clos_name*] [–s *sourcepath* [–iy [–f *sep*]]] [–u *user*] –k *key* –p *pathname*

collector [–C *clos_name*] [–s *sourcepath* [–iy [–f *sep*]]] [–u *user*] –k *key* –c *pathname*

collector –?

DESCRIPTION

The collector command starts a daemon process on the local node. This process sets up a marked channel for every integrated node, and then waits until an originator process sends data on this marked channel.

The Collector daemon can operate in two ways depending on whether the –p option (file mode) or the –c option (command mode) has been set.

In the case of file mode, incoming data is added to the file assigned the *pathname* argument. If the attempt to add data to the file fails, a high-priority message is sent on the relevant channel to the sender of the data, and the Collector daemon terminates.

In command mode, the command with the pathname *pathname* of the –c option is executed when a message is received. The message is the only argument passed to the command.

A Collector daemon can be started several times on each node for the same event class. However, the same result file may not be used more than once in file mode. If another daemon is started with the same target file in file mode, the daemon terminates immediately with an error. This is because the incoming messages would otherwise appear several times in the file and would appear muddled as they are not written to the file in mutually exclusive mode.

The Collector daemon is terminated with the event_stop(1M) command or the local event_adm –D –c command and is sent the SIGTERM signal. An abort condition now exists for the daemon. If an event message is present, which could not yet be written to the *pathname* file, but one of the abort conditions exists, the daemon sends a high-priority message back to the channel on which it received the message before it terminates. An abort condition exists and the Collector daemon terminates if:

- no communication channel can be configured when starting the daemon
- a Collector daemon already exists in file mode with the same *key* and *pathname* arguments
- the daemon is forcibly terminated with the SIGTERM signal
- the last sender closed its channel
- the daemon process failed when writing to the *pathname* file or if it receives a SIGPIPE signal

When the Collector daemon aborts, all related Emitter daemons are also terminated indirectly.

This command is part of the CENS subsystem and can be used in conjunction with the emitter(1M) command which is used to send messages.

Thus, with the help of the –s option, the Collector daemon can have an appropriate Emitter daemon started on every node. The type of event or group of events is transparent for the Collector daemon. This is specified by the filename and the *key* as described in cens(7) . The –s command is used by the event_start(1M) command.

If a node is not to be integrated in the cluster when the Collector daemon is started, or if a new node is to be added during event monitoring, the Collector daemon automatically includes this node in its communication and starts an Emitter daemon if necessary.

OPTIONS

−C *clos_name*

The name of the cluster is selected in the *clos_name* argument with this option. If this option is not specified, the name from the `$CLOSNAME` environment variable is used, or if this is not set, the name is taken from the `/var/clos/.CLOSdefault` file.

−c *pathname*

The −c option refers to a command file when used in conjunction with the *pathname* argument. This command is called every time the `Collector` daemon receives a message. The message itself is the only argument passed to the command.

A pathname must be specified. If this option is used more than once, only the first specification is accepted. Simultaneous use of the −p option is not permitted.

−f *sep*

Setting a separator (field separator).

If the −i (or −y) option was selected to prefix the node tag name (or hardware serial number) before every message, the −f option comes in useful: A separator is then added to the identification. The *sep* argument dictates which character is to be used as the separator.

−i Inserting the tag name.

Every event message consists of a string that ends with a newline character (0x0a). The function of this option is to enable the daemon to prefix every event message with the node ID (tag name) of the local node (followed by a separator). The default separator is the colon (:).

−k *key*

Key for marking the communication channels.

The communication channels to the integrated nodes are marked with the *key* argument so that, for example, all `Emitter` daemon event messages with the same flag can be processed.

A key must be specified. If this option is used more than once, only the first specification is accepted.

−p *pathname*

This option is used to set the file mode. The *pathname* argument represents the pathname to the event file to which the event messages are added. If the attempt to add event messages to the file fails, an abort condition exists and the last message is returned to the sender as not accepted.

A pathname must be specified. If this option is used more than once, only the first specification is accepted. Simultaneous use of the −c option is not permitted.

−s *sourcepath*

After a channel marked with *key* has been configured for every integrated node, the `Collector` daemon starts an `Emitter` daemon on each integrated node to which it passes the *sourcepath* argument. This also occurs on nodes that are integrated subsequently.

This option can be used to configure a process network for certain event messages.

−u *user*

This option determines the user rights for the target file. In conjunction with command mode (−c), the execution rights are set to those of the *user* user. In the case of file mode (−p option), access rights are checked with respect to the *user* user. When files are being created, the *user* user is deemed to be the owner of the respective files.

−y Inserting the hardware serial number.

Every event message consists of a string that ends with a newline character (0x0a). The function of this option is to enable the daemon to prefix every event message with the hardware serial number of the local node (followed by a separator). The default separator is the colon (:).

−? Usage of this command.

DIAGNOSTICS

The `Collector` daemon keeps a log file called `log_`*key* in the `/var/clos/`*clos_name*`/collector` directory. Every entry in this file contains the date, time and process number. The stored information includes when the daemon started, when communication to the nodes was initiated, error situations and when the process terminated along with the reason for the abort.

If the `collector` command comes across an error situation before the daemon process was started, this is reported and no action is performed. The return value is 0 if no error was discovered.

Error messages are sent to standard error output with the following values:

1 "`Illegal arguments`". Incorrect call.

2 "`ERROR on node channel`". Problem with system calls on the channels to the sending nodes.

3 "`poll system call failed`". Problem with the `poll(2)` system call.

4 "`trouble with pathname`". The event file cannot be accessed.

5 "`node not integrated`". The node on which the `Collector` daemon was started is not (no longer) integrated.

6 "`daemon already started pid file locked`". A daemon process with this *key* argument is already active on the same target file.

7 "`selected user not configured`". The *user* user is not known on this node.

If the `Collector` daemon comes across an error, it can no longer report this to standard error output since it no longer has a controlling TTY. The accompanying error message is therefore written to the `log_`*key* file under `/var/clos/`*clos_name*`/collector`. The date, time and daemon process ID are also prefixed to the message text and the daemon then terminates. If the `-c` option is used, the command writes to the `/var/clos/errmsg/cens_cmd` file by default.

FILES

`/var/clos/`*clos_name*`/nodes/`*tag_name*`/collector.`*pid_key*

`/var/clos/`*clos_name*`/collector/log_`*key*

`/var/clos/errmsg/cens_cmd`

SEE ALSO

`emitter(1M)`, `event_adm(1M)`, `event_start(1M)`, `event_stop(1M)`, `poll(2)`, `cens(7)`, `clos(7)`.

Fujitsu Siemens Computers Add-on Reference Manual

ctrldump(1M)

NAME

`ctrldump` - record a controller dump

SYNOPSIS

`/opt/diag.d/bin/ctrldump` –b *boardtype* –o *output* [–n *boardno*]

`/opt/diag.d/bin/ctrldump` [–h]

`/opt/diag.d/bin/ctrldump` [–?]

DESCRIPTION

The `ctrldump` command is used to write a memory dump of the current status of an addressed board to a diagnostic file *output*. Since the memory dump is board-specific, `ctrldump` must be provided separately for each controller type. The –h option is used to query for which controller the `ctrldump` command is being implemented. The structure of the dump returned by `ctrldump` depends on the board and is stored in the diagnostic file as a sequence of *ctrldump_info* parts.

Each `ctrldump` part in the diagnostic file contains a probe part, thus the diagnostic file consists of *Probe* and *Dump* parts. Errors in `probe(1M)` or `ctrldump` are indicated in the diagnostic file header.

OPTIONS

–b *boardtype*

Refers to the `controller board`. In this regard, specification of the board type is requested as an argument. The argument *boardtype* is identical to the string which the `autoconf(8)` command also displays without the board number.

The names of board types which support the `ctrldump` command are permitted (`E_SYSTEM`, if the board type is not supported). You can ascertain which controller board types are permitted by using the –h option.

Specification of the board type is mandatory.

–o *output*

Refers to the diagnostic file to which the data provided by `ctrldump` is written. It is important to note that the command does not check whether there is sufficient space in the file system where this file is located.

Specification of this file is mandatory.

–n *boardno*

Refers to the logical board number in the argument *boardno*. The `ctrldump` command does not check whether a board with this number exists. A *Probe* and a *Dump* part is always generated, which can (both) be empty.

If this option is not used, a *Probe* and a *Dump* part is generated for each logical board that is located on the system. The logical board numbers are recorded in the relevant part headers.

The number of `Probe` parts is entered in the diagnostic file header `dh_pparts`, the number of `Dump` parts is entered in `dh_dparts` and the logical board numbers are marked in `dh_boardno` bit-by-bit.

–h This argument is used to display on which boards or components the `ctrldump` command can be used. The output is sent to standard output `stdout`.

If this option is used, all other options are irrelevant.

–? This argument is used to display the user interface of the `ctrldump` command on `stderr`. The return value is 0.

DIAGNOSTICS

If no errors were detected while the program was running, the program is terminated with 0. Error messages

are output to stderr. The following error codes are provided:

E_ILLOPT	1	illegal user interface on command
E_ILLARG	2	illegal arguments used
E_PERM	3	permission denied to uid
E_CMDDIR	4	No stat on components directory
E_SYSTEM	5	Can't call sh-command
E_LIST	6	List command with error
E_OUTPUT	7	Can't work with output file
E_FILE	8	Can't work with input file
E_MAGIC	9	wrong magic number

NOTES

When the ctrldump command is called, applications can be aborted or even written to a core, as a dump always causes the board or boards to be reset. It is therefore advisable to settle the applications beforehand in such a way that inconsistent data does not occur.

FILES

/opt/diag.d/components
/usr/include/sys/diag.d/diaghead.h

SEE ALSO

probe(1M) , autoconf(8) .

diagdispl(1M) - lan_host(1M)

diagdispl(1M)

NAME

diagdispl - display the diagnostic file

SYNOPSIS

/opt/diag.d/bin/diagdispl [–b [H]] [–p] [–s] [–d] [–a *sequence*] [–n *boardno*] *diagfile*
/opt/diag.d/bin/diagdispl [–h]
/opt/diag.d/bin/diagdispl [–?]

DESCRIPTION

The command displays the diagnostic file header, the configuration part followed by the display of the part types in the file to stdout. The formats of the diagnostic header and the part header are located in diaghead.h.

If there are no suitable commands in /opt/diag.d/components for displaying the individual parts, only the header of any part is presented.

If none of the options –p, –s and/or –d is used, all parts including the part header of the diagnostic file are displayed. The options –p, –s and/or –d are used to filter out only the associated parts including the part header If a suitable command is not available for displaying the part, only the part header is presented.

If the –b option is used, the data is removed from the files in binary format and is written to stdout. The headers are only displayed in stdout when the –b option is given the argument H. The –b option is used as a

filter for binary data.

OPTIONS

`-b` Only binary data is taken from the diagnostic file without the header. All header information is displayed only on `stdout` if the option has the argument `H`. This option can be used to filter individual parts out of the diagnostic file without the header.

`-p` This option causes all *Probe* parts to be filtered out of the diagnostic file and to be displayed. If the developer has not provided a routine for displaying the Probe parts, only the Probe header or headers are displayed. The `-b` option is used to output the Probe parts in binary format. The part header is only output if the argument `H` has been set for the `-b` option.

`-s` This option is used to display all *statistic* parts of the diagnostic file. If the developer has not provided a routine for displaying the statistic parts, only the statistic header or headers are displayed. The `-b` option is used to output the statistic parts in binary format. The part header is only output if the argument `H` is set for the `-b` option.

`-d` This option is used to display all *Dump* parts of the diagnostic file. Since there are no routines for displaying dumps, only the Dump header or headers are displayed. The `-b` option is used to output the Dump parts in binary format. The part header is only output if the argument `H` has been set for the `-b` option.

`-a` *sequence*
This option is used to select and display in accordance with the other options, only that part marked in the part sequence with the number *sequence*. If the argument *sequence* is greater than the number of parts, this number is ignored. This number is also ignored if more than one part type was selected. A corresponding message is displayed on `stderr`.

`-n` *boardno*
This option is used to output only those parts of the diagnostic file to `stdout`, which refer to the logical board number of the argument *boardno*. The other selection criteria depend on the other options.

`-h` This option displays help messages that will enable the user to recognize for which components and part types explicit display routines are available on the respective system.

If this option is used, all of the other options are irrelevant.

`-?` This option is used to display the user interface of the command on `stderr`. The return value is 0.

If this option is used, all of the other options are irrelevant.

diagfile
is the diagnostic file which is displayed. The message is displayed on `stdout`. If errors occur when individual parts are being output, these parts are skipped, but a message is output to `stderr`.

DIAGNOSTICS

If no errors were identified while the program was running, the program aborts with 0. The error messages are output to `stderr`. The following error messages are provided:

`E_ILLOPT`	1	illegal user interface on command
`E_ILLARG`	2	illegal arguments used
`E_PERM`	3	permission denied to uid
`E_CMDDIR`	4	No stat on components directory
`E_SYSTEM`	5	Can't call sh-command
`E_LIST`	6	List command with error
`E_OUTPUT`	7	Can't work with output file
`E_FILE`	8	Can't work with input file

E_MAGIC 9 wrong magic number

FILES

/opt/diag.d/components
/usr/include/sys/diag.d/diaghead.h

Fujitsu Siemens Computers Add-on Reference Manual

diaghead(1M)

NAME

`diaghead -` display the diagnostic file

SYNOPSIS

`/opt/diag.d/bin/diaghead` *diagfile*

DESCRIPTION

This is a command that displays the disagnostic file header located in the `diagfile` file in readable form and then shows the configuration part on `stdout`.

diagfile
is the diagnostic file that is displayed. The display is made to `stdout`.

DIAGNOSTIC

The return value is 0 if an error is not recognized and in an error situation an error message is output to `stderr` and the return value is not equal to 0:

`E_ILLARG`	2	illegal arguments used
`E_PERM`	3	permission denied to uid
`E_FILE`	8	Can't work with input file
`E_MAGIC`	9	wrong magic number

FILES

`/usr/include/sys/diag.d/diaghead.h`

Fujitsu Siemens Computers Add-on Reference Manual

draidd(1M)

NAME

`draidd -` a daemon that monitors both channels of a RAID box

SYNOPSIS

`draidd [ [–D [–cCru ] [ –M` *number* `]]`

`draidd [ –gGHsS ]] [ –lLv ] [ –x` *number* `] [ –t` *file* `] [ –f` *configuration-file* `] [` *config-device* `]`

`draidd [ –a ] [ {–e|–d|–h} ]` *draid-device*

DESCRIPTION

`draidd` (Dual RAID Channel Daemon) is usually started by means of `init(1M)` when the system is booted. `draidd` reads its configuration information from the *configuration-file* at startup; the default name for this file is `/etc/draid`. See also `draid(4)` for the format of the file.

`draidd` creates a redundant access to a RAID box with two controllers which are connected to two host controllers. The daemon uses a system driver which allows access to the RAID box over one or both controllers.

If a controller cannot be accessed, `draidd` reconfigures the routing table of the system driver, and consequently all further access to the RAID box takes place over the other controller.

`draidd` rereads its configuration file as soon as it receives a hangup signal (`SIGHUP`). New RAID box configurations can thereby be activated and existing RAID box configurations can be deleted or modified. The new configuration can be used as soon as the RAID box has been configured successfully.

OPTIONS

`draidd` can be started as a daemon (`–D` option) or `draidd` can be used to read the actual configuration.

The following options can only be used if `draidd` should be started as a daemon:

`–C` Uses and configures the RAID box, as described in the *configuration-file*.

`–c` Uses the current configuration of the connected RAID box (default) until a `SIGHUP` signal is received.

`–D` Becomes a daemon. This flag must only be specified for the `draidd` process which is intended to monitor the RAID boxes. Only one process per system may be started using this flag.

`–M` *number*
: Sets the maximum number of reconfiguration attempts (default: `1`). Once the maximum number of reconfiguration attempts has been reached, access to the entire RAID box is deactivated.

`–r` Creates a "SCSI bus reset" before reconfiguring a RAID box, should a controller fail.

It is recommended that you do not initiate a "bus reset". This flag should only be used in cases where it is clear what is meant by a "bus reset". Data for other SCSI devices on this bus may be lost.

`–u` Updates the `autoconf`-tree after reconfiguration. By default this is switched off. If `–u` is set a reconfiguration may take a longer period of time.

The following options can only be used if `draidd` should not be started as a daemon:

`–G` Displays the RAID box configuration in detail.

`–g` Displays the configuration of the RAID box.

`–H` A hangup signal (`SIGHUP`) is sent to the active daemon.

`–S` Displays statistical data, also for asynchronous tasks.

`–s` Displays statistical data.

Options for high-availability configurations:

A special `draid` device (e.g. `draid0`) must always be nominated for these options.

–a Log information is written to a file on completion of all activities. The file name is made up of the name of the draid device and the process ID.

–h The specified draid device is reconfigured.

The following options are only supported by EMC RAID systems.

–e The SCSI commands sent to the ports of the RAID systems write protect the LUNs of the port to prevent them being accessed from ports other than those connected to this host.

–d Access to the LUNs of the draid device is locked.

The following options are ever valid:

–f *configuration-file*
Uses the *configuration-file* as a configuration description. (default: /etc/draid) [see draid(4)].

–L Output from draidd is not logged with syslogd(1M) , but is sent to the system console. This is the default setting.

–l Logs output from draidd using syslogd(1M) . This option may cause a shutdown in high-load systems when an error occurs, because a file system can no longer be accessed.

–t *file*
Instead of using syslogd(1M) the file *file* will be used for the programs output. This is useful, if the output of draidd starts very early during system startup, where syslogd(1M) is not available.

–v Outputs the program segment where draidd is found. The volume of output increases with each occurrence of –v. syslogd(1M) is used to output the information.

–x 1|2|3
Sets the debug level for the system driver. This should only be used for testing or debugging.

config-device
To enable the required configuration to be loaded in the system driver [see draid(4)], the device nodes can be specified as *config-device*. If no *config-device* is specified during configuration, /dev/draid/draid0l0s0 can be used. This device node is only used to enable communication with the system driver; the nodes are configured as described in draid(4) .

This device node must be specified as *config-device* to enable reading-in of the statistical data of a device node.

SIGNALS

draidd can be affected during runtime by the issuing of signals. Apart from the \SIGHUP signal, all other signals should only be used for debugging or testing.

SIGHUP
draidd reconfigures the RAID box. –H option.

SIGUSR1
Output is issued indicating the program part in which draidd is contained. The volume of output is increased every time SIGUSR1 is sent. –v option.

SIGUSR2
The volume of output is reduced every time SIGUSR2 is sent. –v option.

SIGPWR
Enables or disables the use of syslog(3C) for output. –l or –L options.

SIGPROF
draidd logs information in an internal buffer. When the SIGPROF signal is issued, this buffer is output on the system console and written to the /tmp/draidd.dump file.

FILES

/etc/draid
Configuration description.

/dev/draid/draid0

This device node is used by `draidd` to communicate with the driver.

`/tmp/draidd.dump`
Internal log information of `draidd` daemon.

`./draid`*X*`.`*PID*
Internal log information of a `draidd` process; *X* is the number of affected `draid` devices.

`/dev/autoconf`
This device node is used to determine the RAID controllers device addresses.

`/dev/draid`
Device node directory.

`/tmp/.draidd_lock`
`draidd` never creates this file, however it does delete it following reconfiguration. This file can be created and tested using an external configuration process.

SEE ALSO

`draid(4)`, `draid(7)`.

Fujitsu Siemens Computers Add-on Reference Manual

dumpfile(1M)

NAME

`dumpfile -` decode and display binary configuration file

SYNOPSIS

`dumpfile` [`–f` *filename*]

DESCRIPTION

The command `dumpfile` outputs a readable dump of the binary configuration file (on standard out). This allows to find out which parameters and blocks have been configured in the currently used configuration file.

OPTIONS

`–f` *filename*

Filename of the binary configuration file. If not specified the binary configuration file name `/opt/observe/konfig/konfig.bin` is used as default for analysis.

EXIT STATUS

`0` `dumpfile` terminated without error.

`1` An error occurred.

SEE ALSO

`observe`(5) .

Fujitsu Siemens Computers Add-on Reference Manual

econ_pw(1M)

NAME

`econ_pw` - enabling/disabling environment controllers

SYNOPSIS

`econ_pw –p` *EIP-PID* [`–a on|off`] [`–m`]

DESCRIPTION

The `econ_pw` command can be used to enable and disable the mains socket of an ECON (Environment Controller). This enabling and disabling procedure is not checked.

The commands are synchronized to prevent errors when modifying the EIP poweron mask, i.□e. if a number of `econ_pw` commands are issued on a system, the next command begins executing when the previous command has ended.

OPTIONS

`–p` *EIP-PID*

The physical ID (EIP-PID) of an ECON, if specified, identifies the ECON on the CAN bus. The EIP-PID is defined by setting the DIP switch on the EIP of the ECON and must be entered in hexadecimal notation (range of values 0 to 3F).

`–a on|off`

This option specifies whether an ECON is to be enabled or disabled. If this option is not used, the value `off` is taken to be the default value.

`on` The option `on` enables an ECON.

`off` The option `off` disables an ECON. The socket is only disabled physically if the ECON has been configured as the master on the CAN bus or if the ECON has been configured as the slave on the CAN bus and has only been enabled by one master.

If the ECON has been configured as a slave on the CAN bus and is then enabled by several masters, the poweroff command simply causes the ECON bit in the poweron mask of the respective system to be set to 0. The socket is not disabled physically however.

The socket is only disabled physically when the poweroff command has been issued by all masters that enabled the ECON in the first place.

`–m` The poweron mask may be modified automatically by the EIP poweron/poweroff command used. If the `–m` option is specified, the poweron mask is read prior to the EIP poweron/poweroff command being called, stored locally and restored when the ECON has been enabled or disabled. This ensures that the enabling and disabling procedure for the ECON will be the same the next time the system is powered on or off.

Default value: the mask is not recovered.

DIAGNOSTICS

The return value of the command is 0 and in case of error, not equal to 0.

FILES

`/sbin/econ_pw`
`/sbin/econ_pw.lock`

Fujitsu Siemens Computers Add-on Reference Manual

emitter(1M)
Cluster Operating System

NAME

emitter - forwarding event messages

SYNOPSIS

emitter [–C *clos_name*] –k *key* –p *pathname* [–t *taglist*] [–iy [–f *sep*]]

emitter –?

DESCRIPTION

The emitter command starts a daemon process on the local node that opens a marked channel for every integrated node. The process then waits until a recipient process (Collector daemon) has opened a channel with the same flag and then switches to the transfer status CONNECTED. In this status, the *pathname* file in relation to the –p option is opened, read line by line, and sent to the nodes on these channels. When the file has been fully read, it is shortened to the length 0 and is changed into a communication pipe (FIFO). *Note*: Processes that still have the file open at this time continue to write to this file unnoticed by the daemon. The Emitter daemon then waits in this FIFO for event messages and forwards them to the nodes if necessary.

If a node in the *taglist* is subsequently integrated while in transfer status, the Emitter daemon also opens a marked channel for it and waits for its CLOS_OPEN message. No event messages are sent on this channel until the CLOS_OPEN message is received.

It is therefore ensured that for every node only one Emitter daemon is active with the same *key* argument. If a second daemon is started with the same argument, it terminates immediately.

The daemon can receive the following high-priority check messages on one of the channels:

CLOS_OPEN
: The receiving side signals the daemon indicating that it has opened its communication channel. When the first message of this kind is received, the Emitter daemon switches to the transfer state and begins communication. If the daemon is already in transfer status, the messages for the relevant channel are duplicated.

CLOS_CLOSE
: The receiving side signals the daemon indicating that it has closed its communication channel. The Emitter daemon terminates when the last recipient signals the end of communication.

An abort condition exists and the Emitter daemon terminates if:

- no communication channel could be configured when starting the daemon
- an Emitter daemon already exists with the same *key* argument
- the daemon process was forcibly terminated with the SIGTERM signal
- the daemon process receives the CLOS_CLOSE message from the last recipient
- communication with the last recipient node was interrupted

The emitter command is part of the CENS Subsystem Event Notification Service [see cens(7)] and is complimentary to the collector(1M) command which acts as a recipient for the messages created by the emitter command. The event_start(1M) command is used to start the CENS subsystem whereas the event_stop(1M) command is used to end the subsystem.

OPTIONS

–C *clos_name*
: The name of the cluster is selected in the *clos_name* argument with this option. If this option is not specified, the name from the $CLOSNAME environment variable is used, or if this is not set, the name is taken from the /var/clos/.CLOSdefault file.

–f *sep*

Setting a separator (field separator).

If the –i (or –y) option was selected to prefix the node tag name (or hardware serial number) before every message, the –f option comes in useful: A separator is then added to the identification. The *sep* argument dictates which character is to be used as the separator.

–i Inserting the tag name.

Every event message consists of a string that ends with a newline character (0x0a). The function of this option is to enable the daemon to prefix every event message with the node ID (tag name) of the local node (followed by a separator). The default separator is the colon (:).

–k *key*

Key for marking the communication channels.

The communication channels to the integrated nodes are marked with the *key* argument so that a collector(1M) command with the same flag can continue processing the events. A key must be specified. If this option is used several times, only the first specification is accepted.

–p *pathname*

Pathname of the *pathname* event file.

The –p option is used to specify the pathname of the event file which is first read and then converted to a FIFO. Before this file is converted to a FIFO, the contents read from the file are deleted. If an application writes data to the file after the Emitter daemon has converted it to a FIFO, it remains stored there but is not recognized by the Emitter daemon. A pathname must be specified. If this option is used several times, only the first specification is accepted.

–t *taglist*

List of node tag names. This list of tag names informs the Emitter daemon of the nodes on which the recipient is to search for its messages. The tag names are separated by commas.

If this option is used, the daemon only waits for recipient processes on the nodes in the *taglist*. If this option is not used, the daemon waits for recipient processes on all nodes in an INTEGRATED state.

–y Inserting the hardware serial number.

Every event message consists of a string that ends with a newline character (0x0a). The function of this option is to enable the daemon to prefix every event message with the hardware serial number of the local node (followed by a separator). The default separator is the colon (:).

–? Usage of this command.

DIAGNOSTICS

The Emitter daemon keeps a log file called log_*key* in the /var/clos/*clos_name*/errmsg directory. Every entry in this file contains the date, time and process number. The stored information includes when the daemon started, when communication with the recipient nodes was initiated, error situations and when the process terminated along with the reason for the abort.

If the emitter command comes across an error situation before the daemon process was started, this is reported and no action is performed. The return value is 0 if no error was discovered.

Error messages are sent to standard error output with the following values:

1 "Illegal arguments". Incorrect call.

2 "ERROR on node channel". Problem with system calls on the channels to the sending nodes.

3 "poll system call failed". Problem with the poll(2) system call.

4 "trouble with pathname". The event file cannot be accessed.

5 "node not integrated". The node on which the Collector daemon was started is not (no longer) integrated.

6 "daemon already started pid file locked". A daemon process with this *key* argument is already active on the same target file.

If the Emitter daemon comes across an error, it can no longer report this to standard error output since it no

longer has a controlling TTY. The accompanying error message is therefore written to the `log_`*key* file under `/var/clos/`*clos_name*`/emitter`. The date, time and daemon process ID are also prefixed to the message text, and the daemon then terminates.

Every `Emitter` daemon distributes its status to all nodes if its status changes. This can be seen on every node in the `/var/clos/`*clos_name*`/nodes/`*tag_name*`/emitter_`*key* file, and is displayed using the `event_adm`(1M) command with the `-L` option.

FILES

`/var/clos/`*clos_name*`/emitter/log_`*key*

`/var/clos/`*clos_name*`/nodes/`*tag_name*`/emitter_`*key*

SEE ALSO

`collector`(1M), `event_adm`(1M), `event_start`(1M), `event_stop`(1M), `poll`(2), `cens`(7), `clos`(7).

Fujitsu Siemens Computers Add-on Reference Manual

event_adm(1M)
Cluster Operating System

NAME

event_adm - controlling the Cluster Event Notification Service subsystem

SYNOPSIS

event_adm [–C *clos_name*] –A *flag* –t *event_class* –k *key* –s *source_path* [–p *dest_path*] [–u *user*] [–m *comment*]

event_adm [–C *clos_name*] –D [–ce [–p *dest_path*]] [–t *event_class*] [–k *key*]

event_adm [–C *clos_name*] –I [–t *event_class*] [–k *key*]

event_adm [–C *clos_name*] –L [–ce] [–t *event_class*] [–k *key*]

event_adm [–C *clos_name*] –R –t *event_class* | –k *key*

event_adm [–C *clos_name*] –S [–t *event_class*]

event_adm [–C *clos_name*] –X *nodelist* –t *event_class* | –k *key*

event_adm –?

DESCRIPTION

The event_adm command is used to control the CENS Cluster Event Notification Service subsystem [see cens(7)].

Local CENS subsystem services are executed using the event_adm command. The effect of this command is limited to one node and only produces a side-effect on other nodes if, for example, a CENS daemon is terminated. The Add (–A) and Remove (–R) commands are associated with making changes to the /etc/default/cens file. After this file has been modified, it is distributed to all currently accessible nodes.

Note: Distribution of the file renders the Cluster Object Database inconsistent on every node.

This command recognizes two different CENS daemons, namely the Collector [see collector(1M)] and the Emitter daemon [see emitter(1M)]. If these daemons are not explicitly selected with the –c or –e option, both types of daemon are always selected.

COMMAND OPTIONS

The following basic functions are executed using event_adm:

–A (Add) – Adding an entry

A new entry is added to the /etc/default/cens file. The command checks whether the *event_class* and the*key* are unique and issues an appropriate message. The *flag* determines the operating mode of the event class entered. Any of the following values are possible here:

p (path)
: The Collector daemon is used in file operating mode. If this flag is set, the –s and –p options must be specified.

c (command)
: The Collector daemon is used in command operating mode. If this flag is set, the –s and –p options must be specified.

r (reserved)
: Only the name and the key are reserved. The –p, –s and –u options are ignored and a "–" is entered at the appropriate position in the /etc/defaults/cens file.

Once the entry has been made in the local file, the file is distributed to all accessible nodes. The renders the CLODB Cluster Object Database [see clodb(7)] inconsistent on every node.

–R (Remove) – Removing an entry

The line with the *key* used in the argument or the *event_class* is deleted from the `/etc/default/cens` file. The command checks whether such an entry actually exists and issues a message in the event of an error. After the entry has been deleted from the local file, this file is distributed to all accessible nodes. This renders the `CLODB` Cluster Object Database [see `clodb(7)`] inconsistent on every node.

`-I` (`Integrate`) – Subsequently starting `Emitter` processes
It can happen that an `Emitter` daemon is no longer running on one or more nodes. This can be because the daemon was terminated with a local `event_adm -D -e` command or an error situation arose. In both cases, the associated `Collector` daemon does not try to restart the `Emitter` daemon. The administrator can now attempt to determine the reason for the termination on the local node and rectify the error situation. Using this command, the administrator can then trigger the `Collector` daemon to start the `Emitter` daemon once again. In contrast with the `event_start(1M)` command, a new `Collector` daemon is not started. Instead, an `Emitter` daemon is started on every accessible node if one was not already started.

The command only works on the local node on which a `Collector` daemon is active for this event class.

`-D` (`Delete`) – Ending local daemon processes
The daemon processes selected on the local node are ended using this command. If no special type is selected (`-c` or `-e` option), both local daemon processes for the selected event classes are ended. When ending an `Emitter` daemon, the `Collector` daemon is informed that the `Emitter` daemon is not to be restarted until this has been explicitly signaled (`-I` option or a new node is included in the cluster group).

Since a single node can have several `Collector` processes for one event class, which can be distinguished by their different target files, the target file of the `Collector` daemon to be ended can be specified using the `-p` option. In this case, only one `Collector` daemon is ended.

Ending a `Collector` daemon can result in all `Emitter` daemons being ended, particularly when the `Collector` daemon ended was the only message collector.

`-L` (`List`) – Displaying the active `CENS` processes
This command is used to list process numbers and the *key* of the `CENS` daemons active on the local node. If no event or key was selected, all daemons are listed. The format is either:

`emitter DAEMON: CLOS=`*clos_name* `KEY=`*key* `PID=`*pid* `NUM_NODES=`*num*

node_tag `UP collectors` *no*
...

or:

`collector DAEMON: CLOS=`*clos_name* `KEY=`*key* `PID=`*pid* `CONTROLLED_NODES=`*num*

tag_name `emitter NOT_AVAIL | UP`
...

If no daemon is active for the selected event class, an appropriate message is output.

As the command lists the processes, it simultaneously induces the defined `CENS` daemons to write their internal status to a file and store this file in the `/var/clos/`*clos_name*`/nodes/`*tag_name* directory under the name `emitter_`*key* or `collector.`*pid_key*. Apart from the date and time, this file contains the connection states between the processes in the `CENS` network for a specific event class.

`-S` (`Show`) – Displaying the entries
The entries in the `/etc/default/cens` file are displayed. If no further argument is specified, all event classes are displayed.

`-X` – Setting a default
Using this default, a `Collector` daemon is started on every node in the *nodelist* the next time the `event_start(1M)` command is called.

OPTIONS

–C *clos_name*
The name of the cluster is selected in the *clos_name* argument using this option. If this option is not specified, the name from the `$CLOSNAME` environment variable is used or, if this is not set, the name is taken from the `/var/clos/.CLOSdefault` file.

–c Choosing the `Collector` daemon.

–e Choosing an `Emitter` daemon.

–k *key*
The *key* argument is used to set keys on the communication channels between the daemons and also to distinguish between the various event classes. The assignment of class names to keys is recorded in the `/etc/default/cens` file.

–m *comment*
The argument for this option is added as a comment to the entry in the `/etc/default/cens` file.

–p *dest_path*
This option is used to set the *dest_path* pathname for the `Collector` daemon target file. If the option is not used with the `Add` command, a "–" is entered in the `/etc/default/cens` file and a target file must be specified each time the `event_start(1M)` command is used.

A special `Collector` daemon can be selected using the `Collector` daemon `Delete` command where there are several `Collector` daemons in this event class.

–s *source_path*
This option determines the pathname of the *source_path* source file for an `Emitter` daemon. This option is ignored in the case of event classes with the operating mode "reserved" (`r` flag of the `–A` option). This option is mandatory with all other operating modes.

–t *event_class*
Class (type) of the events.

This option is used to specify the type or class of the event messages. The *event_class* argument, i.e. the name of the event message class (type), is converted as described in `cens(7)` in the `/etc/default/cens` file.

–u *user*
With this command, the name is entered from which the user rights for the target or command file can be derived. If this option is not selected, the user name from the current shell session (`$LOGNAME`) is used.

–? Usage of this command

DIAGNOSTICS

The return value is 0 if no error was discovered. If the `event_adm` command comes across an error situation, this is reported to standard error output and the return value is a value other than 0.

1 "`You must select command option exclusively!`" Incorrect call: Either several or no command options were used.

2 "`Parameter` *arg* `missing!`". Incorrect call: The specified *arg* argument is missing from the command.

3 "`Either` *class* `or` *key* `necessary!`". Incorrect call: Either an event class (`–t` option) or a flag (`–k` option) must be specified for the selected command.

4 "`You can use t/k option only once!`". Incorrect call: The `–t` or `–k` option was used more than once; this is not permitted for the selected command.

5 "`You must select` *class* `or` *key* `argument!`". Incorrect call: An event class (`–t` option) or a flag (`–k` option) must be specified for the selected command.

6 "`add entry => absolute` *source*/*dest* `path requested!`". Incorrect call: The pathname to the *source* source file or the *dest* target file must be absolute.

7 "no such *key*/*class*!". Incorrect arguments: The class or nominated key does not exist.

8 "missing /etc/default/cens file!". The CENS subsystem is not installed correctly.

9 "add entry => *class*/*key* already exists!". Incorrect arguments: The class or nominated key already exists.

10 "No emitter or collector daemon running!". No daemon can be displayed.

NOTES

When using the –A and –R options, the /etc/default/cens file is modified. This modification only occurs at a local level and, apart from any effect on any local CENS daemons, it does not affect the cluster as a whole. However, since the file is part of the CLODB Cluster Object Database, it becomes inconsistent after the event_adm command is executed and should be adjusted [see cod_sync(1M)] so that the same event classes are known on all nodes.

FILES

/etc/default/cens

/var/clos/*clos_name*/nodes/*tag_name*/collector.*pid_key*

/var/clos/*clos_name*/nodes/*tag_name*/emitter_*key*

SEE ALSO

cod_sync(1M), collector(1M), emitter(1M), event_start(1M), event_stop(1M), cens(7), clodb(7), clos(7).

Fujitsu Siemens Computers Add-on Reference Manual

event_start(1M)
Cluster Operating System

NAME

`event_start -` starting the Event Notification Service subsystem

SYNOPSIS

`event_start` [`–C` *clos_name*] [`–iy` [`–f` *sep*]] [`–c`] [`–e` *taglist*] [`–p` *dest*] *event_class*

`event_start –?`

DESCRIPTION

This command is used to start the `Emitter` [see `emitter(1M)`] and `Collector` daemons [see `collector(1M)`] in the Cluster Event Notification Service subsystem `CENS` that are required for a specific class of event [see `cens(7)`].

The command creates an `Emitter` daemon on every integrated node and the associated `Collector` daemon on the current node if other nodes have not been preset using the `event_adm(1M)` command and the `–X` option. The command ensures that every node integrated subsequently is included in the connecting network of these daemons. This means that an `Emitter` daemon is started on a node subsequently integrated and the `Collector` daemon automatically recognizes the new node.

The *event_class* argument, i.e. the name of the event message class, is converted as described in `cens(7)` in the `/etc/default/cens` file.

OPTIONS

`–C` *clos_name*
: The name of the cluster is selected in the *clos_name* argument using this option. If this option is not specified, the name from the `$CLOSNAME` environment variable is used or, if this is not set, the name is taken from the `/var/clos/.CLOSdefault` file.

`–c` Starting a local `Collector` daemon.

: The arguments arising from the other options are used in order to start a local `Collector` daemon. No `Emitter` daemons are started using this daemon with the result that the `–i` (or `–y`) and `–f` options are ignored.

`–e` *taglist*
: Starting a local `Emitter` daemon.

: The arguments arising from the other options are used in order to start a local `Emitter` daemon. A *taglist* list must be specified as the argument for the `–e` option. This is a list of tag names separated by commas. An `all` character string is also permissible as the *taglist*. In this case, an `Emitter` daemon is started which expects the `Collector` processes on all nodes. The `Emitter` daemon to be started sends its event messages to the nodes on this list if a `Collector` daemon has signed on there.

`–i` Inserting the tag name.

: This option is specified with the `Emitter` daemons started for the `CENS`. These daemons insert the tag name of the local node along with a separator (see `–f` option) before every event message. The default separator is the colon (`:`). If no `Emitter` daemon is started, this option is ignored.

`–f` *sep*
: Setting a field separator.

: If the `–i` (or `–y`) option was selected to prefix the node tag name (or hardware serial number) before every message, the `–f` option comes in very useful: A separator is then added to the identification. The *sep* argument determines which character is used as the separator. Only the first ASCII character of the argument is used in this case.

`–p` *dest*

Selecting a target file.

An entry is created in the `/etc/default/cens` configuration file when an *event_class* is specified. This contains the pathname to the file (FIFO) to which the `Collector` daemon writes the accumulated messages. The pathname contained in the `/etc/default/cens` file is overridden with the `-p` option. If the event entry does not contain a target path name and if the `-p` option is not used, an error message (13) is issued.

The pathname for the target file must be absolute.

`-y` Inserting the hardware serial number.

This option is specified with the `Emitter` daemons started for `CENS`. These daemons insert the hardware serial number of the local node together with a separator (see `-f` option) before every event message. The default separator is the colon (`:`). If no `Emitter` daemon is started, this option is ignored.

`-?` Usage of this command.

DIAGNOSTICS

After the `CENS` daemons belonging to a class have been started for every node, an appropriate message is sent to standard error output. Once an error has been detected, the command is aborted and the error reported.

The return value is 0 if no error is discovered. If the `event_start` command comes across an error situation, this is reported to standard error output and the return value is a value other than 0.

`1` "`event class argument required`". Incorrect call: An event class must be specified as an argument.

`2` "`only one event class allowed`". Incorrect call: Only one event class may be started.

`3` "`either e or c option allowed`". Incorrect call: Only a `Collector` or an `Emitter` daemon can be started locally.

`4` "`absolute destination path with p option required`". Incorrect call: An absolute pathname must be specified when using the `-p` option.

`5` "`Can't start notification service for '`*event_class*`'`". Situation error: An attempt was made to start an event class that was not configured for the Event Notification Service subsystem [see `cens`(7)].

`10` "`this node is not INTEGRATED`". Situation error: The local node is not integrated in the cluster.

`11` "`missing /etc/default/cens file! CENS installed?`". Situation error: The `/etc/default/cens` file could not be found.

`12` "`no such event class`". Situation error: The specified event class is not entered in the `/etc/default/cens` file.

`13` "*event_class*`: You must select destination path`". Situation error: No default target file is entered for this event so that one must be specified when using the `-p` option.

`14` "`Couldn't start local emitter Daemon [`*exit_value*`]`". Situation error: The local `Emitter` daemon could not be started. The reason was sent to standard error output before this message was issued or can be found in the accompanying log file.

`15` "`Couldn't start local collector Daemon [`*exit_value*`]`". Situation error: The local `Collector` daemon could not be started. The reason was sent to standard error output before this message was issued or can be found in the accompanying log file.

`16` "`CLOS cluster: '$CLOSNAME' not available!`". Situation error: The name of the cluster cannot be determined. The `/var/clos/.CLOSdefault` file is missing.

`17` "`No such CLOS cluster: '$CLOSNAME' configured!`". Situation error: The specified cluster does not exist.

FILES

```
/etc/default/cens
/var/clos/.CLOSdefault
```

SEE ALSO

collector(1M), emitter(1M), event_adm(1M), cens(7), clos(7).

Fujitsu Siemens Computers Add-on Reference Manual

event_stop(1M)
Cluster Operating System

NAME

`event_stop -` ending the Event Notification Service subsystem

SYNOPSIS

`event_stop` [`–C` *clos_name*] *event_class* [*event_class* ...]

`event_stop –?`

DESCRIPTION

The `event_stop` command is used to clear down the connecting network between the `Emitter` and `Collector` daemons, which were started for an event class with the `event_start`(1M) command, and terminate the processes. In contrast with the `event_adm –D` command, this command works throughout the cluster, i.e. it is guaranteed that all active `CENS` daemons are terminated for this event class.

The *event_class* arguments, i.e. the names of the event message classes, are converted as described in `cens`(7) in the `/etc/default/cens` file.

OPTIONS

`–C` *clos_name*
: The name of the cluster is selected in the *clos_name* argument using this option. If this option is not specified, the name from the `$CLOSNAME` environment variable is used or, if this is not set, the name is taken from the `/var/clos/.CLOSdefault` file.

`–?`
: Usage of this command.

DIAGNOSTICS

After the `CENS` daemons belonging to a class have been ended for every node, the message:

"`event class '`*event_class*`' stopped`"

is sent to standard error output. After an error has been detected, the command is aborted and the error reported.

If a reserved key is specified in the *event_class* list, this class is ignored and the message:

"`Class '`*event_class*`' is a reservation for key` *event_class*"

is sent to standard error output.

The return value is 0 if no error is detected. If the `event_stop` command comes across an error situation, this is reported to standard error output and the return value is a value other than 0.

`1`
: "`class argument required`". Call error: At least one event class must be specified.

`2`
: "`only one event class allowed`". Call error: Only one event class can be terminated.

`10`
: "`Can't assign CLOS name`". A cluster name cannot be determined. The `/var/clos/.CLOSdefault` file does not exist.

`11`
: "`missing /etc/default/cens file!`". Situation error: The `/etc/default/cens` file could not be found.

`12`
: "`no such event class`". Situation error: The specified event class is not entered in the `/etc/default/cens` file.

`16`
: "`CLOS cluster` *clos_name* `not available!`". Situation error: The local node is not integrated in the cluster.

`17`
: "`No such CLOS cluster '`*clos_name*`' configured!`". Situation error: The specified cluster is not configured.

FILES

/etc/default/cens
/var/clos/.CLOSdefault

SEE ALSO

collector(1M), emitter(1M), event_adm(1M), event_start(1M), cens(7), clos(7).

Fujitsu Siemens Computers Add-on Reference Manual

fstyp(1M-hs)

NAME

`fstyp(hs)` - determine file system type

SYNOPSIS

`fstyp` [`–v`] *special*

DESCRIPTION

`fstyp` allows the user to determine the file system type of unmounted file systems using heuristic programs.

An `fstyp` module for each file system type to be checked is executed; each of these modules applies some appropriate heuristic to determine whether the supplied *special* file is of the type for which it checks. If it is, the program prints on standard output the usual file system identifier for that type and exits with a return code of 0; if none of the modules succeed, the error message `unknown_fstyp (no matches)` is returned and the exit status is 1. If more than one module succeeds the error message `unknown_fstyp (multiple matches)` is returned and the exit status is 2.

If it is a High Sierra files system the string `hs` is returned.

OPTIONS

`–v` Produce verbose output, displays various information about the CD-ROM.

Example:

```
fstyp –v /dev/ios0/rsdisk006s0
```

produces:

```
hs
     volumeid    testdisk
     systemid    Siemens Nixdorf Informationssysteme AG
     type        ISO0960
     lbsize      2048
     volspacesz  258788
     volsetsz    1
     volseqno    1
     ctime       Thu Jul 20 15:42:31 1989
     mtime       Thu Jul 20 15:42:31 1989
```

NOTES

The use of heuristics implies that the result of `fstyp` is not guaranteed to be accurate.

SEE ALSO

`hs`(7) .

Fujitsu Siemens Computers Add-on Reference Manual

get_uname(1M)
Cluster Operating System

NAME

`get_uname` - get the node name of an IP address in the cluster

SYNOPSIS

`get_uname` [`-C` *clos_name*] [`-c` *name*] [`-i`] [`-t` *sec*] [`-v` *string*] *hostname*

`get_uname -?`

DESCRIPTION

Node names are converted using the `get_uname` command.

A request for the node name is sent to the *hostname* address. This query is directed to the Cluster Service daemon [see `clserver(1M)`] installed there. Using with `uname(2)` function, the Cluster Service daemon determines the node name and then returns it.

The node name of a node can only be queried if the password for the superuser of the new node is known. The encrypted password can be transferred with the `-v` option. If this option is not used, the password is queried interactively.

The `get_uname` command sends the name to standard output.

OPTIONS

`-C` *clos_name*
: The name of the cluster is selected in the *clos_name* argument using this option. If this option is not specified, the name from the `$CLOSNAME` environment variable is used or, if this is not set, the name is taken from the `/var/clos/.CLOSdefault` file.

`-c` *name*
: Comparing names.

When this option is used, the *name* argument is compared with the node name identified. If both names are the same, 0 is returned, whereas 1 is returned if the names differ.

`-i` The IP address is sent to standard output in addition to the output of the node name.

`-t` *sec*
: Adjusting the timeout.

Since it can happen that the preset *hostname* address may not be accessible when `get_uname` is called (because, for example, the node was disabled), the command is aborted by default with error no. 4 after 15 seconds have elapsed. This period of time can be adjusted using this option where the *sec* argument specifies the time in seconds until the command is aborted.

`-v` *string*
: The string transferred with this option is sent as an encrypted superuser password for the addressed node. The superuser (root) password is then checked to ensure it is correct.

If this option is not used, the password is established interactively. The command aborts if the password is incorrect.

`-?` Usage of this command.

DIAGNOSTICS

If the `get_uname` command was successful, the value 0 is returned. If system calls are aborted, their messages are also sent to standard error output as error text. The following return values as possible:

`0` The node name was determined and sent to standard output. When compared, the node name was the same as the *name* argument of the `-c` option.

1 The node name was determined and sent to standard output. However, when compared with the *name* argument of the `-c` option, the names were shown to be inconsistent. This means that *name* is not the name of the node that can be accessed via the *hostname* address.

2 "`Can't contact host`". The node was unable to respond within the specified period of time (default 15 seconds).

3 "`Can't create receive socket`". Problem on the local node: No communications endpoint could be created.

4 "`Can't send RQ_UNAME to host`". Problem on the local node: The specified node could not be accessed.

5 "`Can't receive response`". Problem on the local node when receiving the response to the query.

6 "`No such CLOS` *clos_name* `on remote host`". A cluster called *clos_name* is not configured on the *hostname* system being sought.

7 "`Tag` *tag_name* `already in a` *clos_name* `CLOS`". The *hostname* system being sought already belongs to a cluster called *clos_name*.

9 "`Host` *tag_name*`: Sorry password incorrect!`". The password sent does not match that of the superuser.

10 "`illegal option`". Incorrect call: An illegal option was selected.

11 "`hostname missing`". Incorrect call: The *hostname* command parameter was not specified.

SEE ALSO

`clserver`(1M), `uname`(2), `clos`(7).

Fujitsu Siemens Computers Add-on Reference Manual

hvassert(1M)

NAME

`hvassert -` assert (test for) Reliant Monitor resource state

SYNOPSIS

Format 1

`hvassert [ –h` *SysNode* `] [ –q ] –s` *resource_name resource_state*

Format 2

`hvassert [ –h` *SysNode* `] [ –q ] –w` *resource_name resource_state seconds*

DESCRIPTION

The `hvassert` command tests for a specified resource state for a Reliant Monitor resource. Can be used in scripts when a resource must achieve a specified state before the script can issue the next command.

OPTIONS

`–h` *SysNode*
: Perform the assertion on *SysNode*. Without this option, the assertion is done on the local system node. This command is equivalent to logging into the named *SysNode* and issuing the assertion.

`–q`
: Hvassert works silently without printing any messages. The exit code alone determines success or failure.

`–s` *resource_name resource_state*
: Assert that *resource_name* is in state *resource_state*. This command tests the current state of the node.

`–w` *resource_name resource_state seconds*
: Assert the resource *resource_name* state is *resource_state*, where *seconds* is the maximum number of seconds to wait before reporting.

  `hvassert –w` works by polling Reliant Monitor once each second until either the asserted condition is true or the specified timeout has elapsed. It is possible for `hvassert` to miss a transient state change if it occurs within a one-second interval between polls.

DIAGNOSTICS

If zero is returned in response to the `hvassert` command, the asserted state is true. If non-zero is returned, the asserted state is not true or the command failed for some reason.

EXAMPLES

The following example asserts that `HOST-1` resource is in the online state:

```
hvassert –s HOST-1 Online
```

SEE ALSO

`hvdisp`(1M).

Fujitsu Siemens Computers Add-on Reference Manual

hvbuild(1M)

NAME

`hvbuild -` build Reliant Monitor nodes configuration file

SYNOPSIS

`hvbuild` *config_file*

DESCRIPTION

The `hvbuild` command builds (compiles) the Reliant Monitor nodes configuration file and generates the corresponding shared library file in the `/usr/opt/reliant/conf` directory.

ARGUMENTS

config_file

Name of configuration file being built. The source file name must have the `.us` extension.

To change a Reliant Monitor configuration, use `hvgen` to regenerate the nodes configuration file or a text editor to modify the current file. Rebuild the nodes configuration file and then restart Reliant Monitor.

In addition to building a new nodes configuration file, two additional files are automatically created when the `hvbuild` command is run: *config_file*`.files` and *config_file*`.nodes`. These files are used by the `hvdist` command to distribute files across the nodes in the configuration.

EXAMPLES

The following command builds the configuration file `sample1.us`:

```
hvbuild sample1
```

SEE ALSO

`hvcm`(1M), `hvdist`(1M), `hvgen`(1M).

Fujitsu Siemens Computers Add-on Reference Manual

hvcm(1M)

NAME

hvcm - start the Reliant Monitor configuration monitor

SYNOPSIS

hvcm [−c *config_file*] [−m] [−h *time*] [−l *level*] [−r *count*] [−w *time*] [−n]

DESCRIPTION

The configuration monitor is the decision-making module of Reliant Monitor. It controls the configuration and access to all Reliant Monitor resources. If a resource fails, the configuration monitor analyzes the failure and initiates the appropriate action according to the specifications for the resource in the nodes configuration file.

The hvcm command starts the following modules which support the configuration monitor: the communications daemon (hvcommds and hvcommdc), the GUI daemon (hvguiD), and the detectors for all monitored resources. In most cases, it is not necessary to specify options to the hvcm command; the default values are sufficient for most configurations.

If the nodes definitions have been divided into multiple configuration files, all files must be named when the hvcm command is issued. The file containing the `include "hvstub.us"` instruction must be specified last, preceded by the −c option.

OPTIONS

−c *config_file*

Specify the nodes configuration file that should be referred to by the configuration monitor. If this option is not specified, hvcm assumes the nodes configuration file is called config. If an absolute path is specified, only that file name is used. If a relative file name is specified, a search is made for the first match to *config_file* that is found in the search paths specified by the environment variable LD_CONF_PATH.

−h *time*

For SMP systems, specify *time* in seconds to delay action for missed heartbeat. The number of retry attempts can be specified using the −r option.

−l *level*

Print out log messages for *level* where *level* is a list of numbers or a range. Individual levels in the list may be separated by commas or whitespace (whitespace in the list requires that the entire argument be enclosed in quotes). A range of levels is given as *n1*-*n2*; this includes all levels from *n1* up to, and including, *n2*. Range -*n2* is the same as 1-*n2*. Range *n1*- specifies all levels starting with *n1*. The value of *n1* must be greater than or equal to 1.

NOTE: Most logging levels are used for debugging and are not of interest to users. Additionally, running Reliant Monitor with several log levels turned on can adversely affect system performance. Valid logging levels are:

0 Turn on all logging levels.

1 Turn on hvcommdc and hvcommds tracing.

2 Turn on detector tracing.

3 Turn on hvguiD tracing.

4 Turn on mskx stack tracing.

5 Error or warning message.

6 Level for heartbeats and communications daemon.

7 Level for configuration monitor.

8 Detector error.

9 Administrative command message.

10 Basic type level.

11 Verify links level.

12 `DerivedFrom()` level.

13 Token level.

14 Detector message.

15 Local queue level.

16 Local queue level.

17 Script level.

18 Contracting script level.

19 Temporary debug traces.

20 SysNode traces.

`-m` Create `htm` log.

`-n` Run Reliant Monitor in non-DLM mode.

`-r` *count*
Specify retry count for `hvdet_node` detector before offline is reported. This detector reports when a `SysNode` type resource is down only after *count* attempts to connect have failed.

`-w` *time*
Set the detector time period to *time* seconds.

EXAMPLES

The following command starts Reliant Monitor with the `sample1` nodes configuration file:

```
hvcm -c sample1
```

The following command starts Reliant Monitor with log levels 1, 2, 3, and 4 enabled using the `sample1` nodes configuration file:

```
hvcm -l 1-4 -c sample1
```

SEE ALSO

`hvassert`(1M), `hvbuild`(1M), `hvdisp`(1M), `hvdist`(1M), `hvgen`(1M), `hvshut`(1M), `hvswitch`(1M), `hvthrottle`(1M), `hvutil`(1M).

Fujitsu Siemens Computers Add-on Reference Manual

hvdisp(1M)

NAME

`hvdisp -` display Reliant Monitor resource information

SYNOPSIS

`hvdisp` {`–a` | `–l` | `–S` *resource* | `–T`*resource_type* | *resource* | `–u`} [`–o` *out_file*]

DESCRIPTION

The `hvdisp` command displays information about the current configuration for Reliant Monitor resources.

OPTIONS

`–a` For each resource in the configuration, display the *resource* name, *resource_type*, `HostName` attribute, and for local resources the current resource state.

`–l` Display the same information as `–a` and in addition show the parse stack for each resource.

`–S` *resource*
Display the same information as `–a`, but only for resource nodes that are descendants of *resource*. This option overrides the `NoDisplay` attribute.

`–T` *resource_type*
Display the same information as `–a`, but only for nodes of type *resource_type*.

resource
Display the current state and definition for *resource*. List its children and parents.

`–u` Display the same information as `–a` and update the states every four seconds if any state has changed. The command continues to run until Ctrl-C is issued or Reliant Monitor is shut down.

`–o` *out_file*
Send the output to a file called *out_file*.

EXAMPLES

The following command sends the type and state of the resource named `HOST-2` to a file called `hvdisp.out`:

```
hvdisp HOST-2 –o hvdisp.out
```

SEE ALSO

`hvassert`(1M), `hvgen`(1M).

Fujitsu Siemens Computers Add-on Reference Manual

hvdist(1M)

NAME

`hvdist -` distribute Reliant Monitor configuration files

SYNOPSIS

`hvdist [ –t | –v ] [ –r ] [ –f`*config_file*`.files ] [ –n` *config_file*`.nodes ]` *config_file*

DESCRIPTION

The `hvdist` command is used to distribute configuration *config_file* script and data files to all nodes within a Reliant Monitor configuration. In order to successfully use the `hvdist` command, two files must exist: *config_file*`.files` and *config_file*`.nodes`.

The *config_file*`.files` file contains pathnames of the files that will be distributed across the nodes of the configuration by the `hvdist` command. This file is first created by the `hvbuild` command. It must then be modified by the administrator to include any script or data file names that might be used when running the Reliant Monitor configuration. By default, this file is named *config_file*`.files`. If this file name already exists, it is not overwritten.

The *config_file*`.nodes` file contains the node names for all the machines in the Reliant Monitor configuration. The *config_file*`.nodes` file is automatically created whenever the `hvbuild` command runs.

By default, the `hvdist` command uses the `rdist(1M)` command for file distribution.

OPTIONS

`–t` Show commands, test pathnames and nodes. No commands are executed when using this option.

`–v` Verbose mode. Show and execute commands.

`–r` Use the `rcp` command instead of the `rdist` command. For large configurations, the `–r` option may be most efficient.

`–f` *config_file*`.files`
Use the *config_file*`.files` for distribution. Default filename is `/usr/opt/reliant/build/`*config_file*`.files`.

`–n` *config_file*`.nodes`
Use the *config_file*`.nodes` for distribution. Default filename is `/usr/opt/reliant/build/`*config_file*`.nodes`.

config_file
config_file specifies the nodes configuration file.

EXAMPLES

The following example creates a *config_file*`.files` called `Source.files` and a *config_file*`.nodes` file called `Source.nodes`:

```
hvbuild Source
```

The following example distributes the contents of the `Source.files` file across the nodes that are specified in the `Source.nodes` file:

```
hvdist Source
```

SEE ALSO

`rcp(1)`, `hvbuild(1M)`, `hvgen(1M)`, `rdist(1M)`.

Fujitsu Siemens Computers Add-on Reference Manual

hvgdmake(1M)

NAME

`hvgdmake` - compile a Reliant Monitor custom detector

SYNOPSIS

`hvgdmake` *detector_name*

DESCRIPTION

The `hvgdmake` command makes (compiles) a custom detector so that it can be used in the Reliant Monitor configuration. The user first prepares a source file for the detector which must be a file with a `.c` extension.

ARGUMENTS

detector_name
Name of source file being converted to a custom detector file.

EXAMPLES

The following command makes a custom detector from the source file `DBdetector.c`:

```
hvgdmake DBdetector
```

SEE ALSO

`hvdist`(1M), `hvgen`(1M), `hvgdstartup`(4).

Fujitsu Siemens Computers Add-on Reference Manual

hvgen(1M)

NAME

`hvgen` - generate a Reliant Monitor nodes configuration file

SYNOPSIS

`hvgen` [`–f`] *highlevel_file*

For SMP systems: `hvgen` [`–f`] [`–d` *dktab_file*] [`–m` | `–M` *cleanup_file*] *highlevel_file*

DESCRIPTION

The `hvgen` command generates a nodes configuration file from a high-level representation of the configuration and a set of standard template files. The time required to produce the configuration file can be shortened by using this method. The low-level configuration generated by the `hvgen` command is placed in a file named *highlevel_file*`.us`.

OPTIONS

`–f` Force overwrite of a writable low-level configuration file.

highlevel_file
: Specify the basename of the high-level input file. The `hvgen` command will look for a file named *highlevel_file*`.hl`.

`–d` *dktab_file*
: Valid for SMP systems only. Use the specified virtual disk file instead of using the system `/etc/dktab` file.

`–m` | `–M` *cleanup_file*
: Valid for SMP systems only. Missing directories and special files will be created using `mkdir`(2) and `mknod`(2), respectively. The commands necessary to restore the system to its original state, `rm`(1) and `rmdir`(1), are placed in the file *cleanup_file*, if specified, or `/var/tmp/Mknod.cleanup`. These options allow debugging or creating nodes configuration files ahead of the hardware.

EXAMPLES

The following command generates the low-level configuration in the file `newconfig.us` from `newconfig.hl`:

```
hvgen newconfig
```

FILES

`/usr/opt/reliant/etc/templates/*`

Valid for SMP systems only:

`/etc/dktab`

`/var/tmp/Mknod.cleanup`

SEE ALSO

`hvbuild`(1M), `hvdisp`(1M), `hvdist`(1M), `hvgdmake`(1M), `hvgdstartup`(4).

Fujitsu Siemens Computers Add-on Reference Manual

hvlogclean(1M)

NAME

`hvlogclean -` clean Reliant Monitor log files

SYNOPSIS

`hvlogclean [–d]`

DESCRIPTION

The `hvlogclean` command saves old log files into a subdirectory whose name is the time Reliant Monitor was last started (unless the `–d` option is used to delete the old log files instead). Regardless, `hvlogclean` creates a clean set of log files even while Reliant Monitor is running.

A nightly `cron` process runs an `hvlogclean` process that deletes any backup log files that are older than the specified deletion time interval. The deletion time interval can be changed by setting the `RELIANT_LOG_LIFE` environment variable.

OPTIONS

`–d` Delete old log files instead of saving them.

EXAMPLES

The following command saves all Reliant Monitor log files to `/var/opt/reliant/log/`*last_cm_start* and establishes new log files:

```
hvlogclean
```

Fujitsu Siemens Computers Add-on Reference Manual

hvshut(1M)

NAME

`hvshut -` shut down Reliant Monitor

SYNOPSIS

`hvshut` {`–f` | `–L` | `–a` | `–l` | `–s`*SysNode*}

DESCRIPTION

The `hvshut` command shuts down the Reliant Monitor software on one or more nodes in the configuration. The configuration monitor on the local node sends a message to other online nodes indicating which node(s) will be shut down. A shutdown warning is also sent to the RCVM (Reliant Cluster Visual Manager) ordering the lower level windows to be closed. The `hvshut` command disables all error detection and recovery routines on the nodes(s) being shut down, but does not shut down the operating system. If any user application nodes are online when the `–f` or `–L` options are used, the applications remain running but are no longer monitored by Reliant Monitor.

The amount of time `hvshut` waits for a successful shutdown is a calculated value based on the size of the configuration. This time interval can be changed using the `RELIANT_SHUT_MIN_WAIT` environment variable.

OPTIONS

`–f` Force a shutdown of Reliant Monitor on the local system. This option forces the configuration monitor to clean up and shut down Reliant Monitor on the local system without performing offline processing.

IMPORTANT: The `–f` option should be used only after a graceful shutdown of the cluster using the `–a`, `–l`, or `–s` option has been attempted. Since the `–f` option skips the offline scripts, if they are required to assure the condition of your resources, they must be manually started.

WARNING: `hvshut –f` may break the consistency of the cluster. No further action may be executed by Reliant Monitor until the cluster consistency is re-established. This re-establishment includes restart of Reliant Monitor on the shutdown host.

`–L` Shut down Reliant Monitor on the local node without shutting down running applications. All Reliant Monitor nodes are notified that a controlled shutdown is under way on the local node. A message displays asking for confirmation before this action begins.

WARNING: `hvshut –L` may break the consistency of the cluster. No further action may be executed by Reliant Monitor until the cluster consistency is re-established. This re-establishment includes restart of Reliant Monitor on the shutdown host.

`–a` Shut down Reliant Monitor on all nodes. An attempt will be made to bring all online user applications offline.

WARNING: If a user application is faulted or offline, no attempt is made to bring its child resources offline.

`–l` Shut down Reliant Monitor on the local node. This option performs offline processing for user applications that are online on the node.

`–s`*SysNode*
Shut down Reliant Monitor on *SysNode*. This option performs offline processing for user applications that are online on the node.

EXAMPLES

The following command shuts down Reliant Monitor on `HOST-1`:

```
hvshut –s HOST-1
```

SEE ALSO

hvcm(1M) .

Fujitsu Siemens Computers Add-on Reference Manual

hvswitch(1M)

NAME

`hvswitch -` switch control of a Reliant Monitor user applicationresource to another system node

SYNOPSIS

`hvswitch` [`–f`] *userApplication* [*SysNode*]

DESCRIPTION

The `hvswitch` command manually switches control of a user application resource from one system node to another in the Reliant Monitor configuration. The resource being switched must be of type `userApplication`. The system node must be of type `SysNode`.

OPTIONS

`–f` Forces the switch even if some underlying resources are faulted or if all system nodes are not online.

userApplication
Specifies the user application resource.

SysNode
Specifies the system node to switch to. If specified, a directed switch of *userApplication* to *SysNode* is made, else a priority switch to the highest priority system node is made.

EXAMPLES

The following command switches the user application `App1` to the system node `HOST-1`:

```
hvswitch App1 HOST-1
```

SEE ALSO

`hvassert`(1M), `hvcm`(1M), `hvdisp`(1M).

Fujitsu Siemens Computers Add-on Reference Manual

hvthrottle(1M)

NAME

`hvthrottle -` prevent multiple Reliant Monitor scripts from runningsimultaneously

SYNOPSIS

`hvthrottle` [`-l` *lock_file*] *command*

DESCRIPTION

The `hvthrottle` command is used within a configuration file to prevent multiple scripts from running at the same time.

OPTIONS

`-l` *lock_file command*

Lock *lock_file*, execute *command*, then unlock *lock_file*. Default *lock_file*: `/usr/opt/reliant/locks/hvthrottle.lock`

EXAMPLES

An example online script might contain the following instruction:

for MPP systems

```
OnlineScript = "hvthrottle fs_on fs_md23 HOST-1 /dir1"
```

for SMP systems

```
OnlineScript = "hvthrottle fs_on rxfs /dev/vd/vdisk23 /dir1"
```

SEE ALSO

`hvswitch`(1M), `hvutil`(1M).

Fujitsu Siemens Computers Add-on Reference Manual

hvutil(1M)

NAME

`hvutil -` manipulate availability of a Reliant Monitor resource

SYNOPSIS

Format 1

`hvutil` {`–a` | `–d` | `–f` | `–c`} *userApplication*

Format 2

`hvutil –t` [*N*] *resource*

Format 3

`hvutil` {`–o` | `–u`} *SysNode*

DESCRIPTION

The `hvutil` command performs the following resource administration tasks: activating or deactivating a resource, bringing a resource offline, clearing faulted resources or hung *SysNodes* in the `Wait` state, and setting detector time periods.

OPTIONS

`–a` *userApplication*

Activate a deactivated *userApplication*. Only use on a user application that has been deactivated with the `–d` option. The `–a` option does not bring *userApplication* online.

`–d` *userApplication*

Deactivate *userApplication*. Because the `–d` option deactivates *userApplication* without removing it from the configuration, it is useful in a situation (such as maintenance) where another node must be prevented from taking control of *userApplication*.

If *userApplication* is online, the `–d` option first brings the application and all its children offline before the application goes to the `Deact` state. The user application cannot be switched until it has been activated with the `–a` option.

`–f` *userApplication*

Bring an online *userApplication* offline without initiating a switchover or shutting down Reliant Monitor. *userApplication* is the name of the user application to be brought offline.

NOTE: To bring *userApplication* back online after using the `–f` option, use the `hvswitch` command.

`–c` *userApplication*

Performs online or offline processing for all children of the specified *userApplication* to attempt to clear any faulted condition. If *userApplication* is offline or faulted, offline processing is performed. If *userApplication* is online, online processing is performed.

`–t`[*N*] *resource*

Set the detector time period for the specified resource to *N* seconds. If *N* is not specified, the time period returns to the default detector time period for that particular kind of detector. *resource* is the name of the resource the detector monitors.

`–o` *SysNode*

Clear the `Wait` state for the specified *SysNode* on all cluster hosts, after the single console (scon) failed to kill the cluster host (*SysNode*), by returning the specified *SysNode* to the online state. If the *SysNode* is currently in the `Wait` state and if the last detector report for the *SysNode* is the online state, the `Wait` state is cleared and the *SysNode* goes back back to the online state, as if no kill request had ever been sent.

This command alleviates the necessity of actually having to kill a cluster host manually to clear a hung cluster, with various *SysNode*s in the `Wait` state, if the reason for the original fault or unexpected offline

can be corrected.

If e.g., the cluster network fails, the cluster machines will no longer recognize each other and will send kill requests to the single console. If these kill requests can not be successfully processed, the cluster will hang with the *SysNode*s remaining in the `Wait` state. No automatic application switchovers will occur. If the network problem can be repaired, the cluster can be reinitialized to the original online state with the `hvutil –o` command. In this case, it is not necessary to actually kill any machines.

`–u` *SysNode*

Clear the `Wait` state for the specified *SysNode* on all cluster hosts, after the single console (scon) failed to kill the cluster host (*SysNode*), by simulating a successful kill message from the scon. This command assumes the cluster host has been manually "killed", i.e. it has been shutdown such that no cluster resources are online. If this command is executed without first having manually "killed" the cluster host, data corruption may occur!

Once this command is executed, the automatic application switchovers will occur just as if the single console had successfully killed the specified cluster host (*SysNode*).

EXAMPLES

The following command deactivates the `App1` user application:

```
hvutil –d App1
```

The following command takes the `App2` user application offline:

```
hvutil –f App2
```

SEE ALSO

`hvassert`(1M), `hvcm`(1M), `hvdisp`(1M), `hvswitch`(1M), `hvthrottle`(1M).

Fujitsu Siemens Computers Add-on Reference Manual

icfmap(1M)

NAME

`icfmap -` list compute node information

SYNOPSIS

`/sbin/icfmap [-l]`

DESCRIPTION

Each compute node in the cluster maintains a node name space that is used to maintain compute node status and internal Interconnect address information. This information reflects the configuration of the Internode Communication Facility (ICF).

The `icfmap` utility lists all of the entries found in the node name space that is maintained on the compute node on which the utility is executed.

The information listed by `icfmap` is formatted as follows:

`nodenumber`
: The first column of output shows the node number for the compute node. It is followed by a "*" sign for the local node.

`nodename`
: This column of output shows the compute node's node name.

`status`
: The current status for the compute node:

  `UP` The ICF on the compute node is configured and operational.

  `DOWN` The compute node was part of the ICF configuration at some time but has been removed for the configuration.

  `CONF` The process of initializing the ICF configuration for the compute node is in progress or has failed.

  `HOLE` A compute node has never been configured to use the node id in the node name space.

  `UNKNOWN`
  : The node name space contains a status indication for the compute node that is not understood by the `icfmap` utility.

`amt entry`
: This column lists the Interconnect IP address configured for the compute node. Each compute node in the cluster maintains an address mapping table (AMT) that provides mapping between Interconnect IP addresses and Interconnect communication addresses. This mapping is the equivalent of ARP mapping for Ethernet.

`eip` This column lists the EIP address configured for the compute node.

The `-l` option causes additional information of each compute node and its Interconnect to be printed.

To retrieve this information privileged operation mode (e.g. super user privileges) is necessary. If an ordinary user supplies this option an additional comment will be printed to point out this lack of privileges.

The information itself gives additional information for each "route" (node address) the compute node has to the Cluster Interconnect. Its format depends on the type of the Interconnect. For an Ethernet Interconnect with two routes an example is:

```
NODE CLUSTER      NAME         STATUS    AMT-ENTRY       EIP
  1  dbcluster    hierro       UP        192.168.0.1     0x00
       na[0]: dev (380,0), ctrl (0,0) addr (e,08:00:06:0b:13:5d), ms E
       na[1]: dev (380,1), ctrl (1,1) addr (e,08:00:06:0b:13:64), ms E
```

The information listed is:

`na[#]`
The number of the node address.

`dev (major,minor)`
The device numbers of the corresponding ICF device.

`ctrl (controller,target)`
The corresponding local ICF controller number and its peer ICF controller number.

`addr (address)`
Gives the address for the Interconnect segment.

`ms state`
Is the current state of this route. Its value may be

`E` for a functional route

`POLL` for a route which is being polled

`NONE` for a defect route

`U` for an unknown state

`n.a.` for a route state which is not available (i.e. the corresponding node is marked DOWN)

SEE ALSO

`nodeinfo(1)`, `arp(1M)`, `nodenum(1M)`, `arp(7)`.

Fujitsu Siemens Computers Add-on Reference Manual

labelit(1M-hs)

NAME

`labelit(hs)` - return volume name for hs file systems

SYNOPSIS

`labelit` [`-F hs`] [*generic_options*] *special* [*fsname volume*]

DESCRIPTION

generic_options are options supported by the generic `labelit`(1M) command.

`labelit` returns the volume name of the file system in device *special*. It can be used for checking the volume identification.

The *special* name should be the physical disk section (for example, `/dev/ios0/rsdisk003s0`). The device may not be on a remote machine.

The *fsname* argument represents the mounted name (for example, `root`, `u1`, etc.) of the file system.

volume may be used to equate an internal name to a volume name applied externally to the disk pack, or tape.

For file systems on disk, *fsname* and *volume* are recorded in the superblock.

OPTIONS

`-F hs`

Specifies the `hs`-FSType. Example:

```
labelit -F hs /dev/ios0/rsdisk003s0
```

produces:

```
volume: Siemens Nixdorf Informationssysteme AG
lbsize: 2048
```

SEE ALSO

generic `labelit`(1M).

lan_host(1M)

NAME

`lan_host -` retrieve information about OBSERVE status

SYNOPSIS

`lan_host` [`–O` | `–B` | `–OB` | `–A`] *host1 host2* [*port*]

DESCRIPTION

This administration command is provided for reasons of upward compatibility. If you work with OBSNET, you do not need `lan_host`.

The `lan_host` command is intended for client-server configurations where the server is duplicated for security reasons and monitored by OBSERVE. The `lan_host` command queries the client as to which of the two computers in the OBSERVE cluster has `O`, `B` or `OB` status. This allows you to stipulate, for example, that it should always be the original system that is accessed via LAN even after a switchover.

The binary code for this command is supplied in the `/opt/observe/bin` directory. In addition, in the `/opt/observe/src` directory is the source code of the command with the file name `lan_host.c`.

OPTIONS

`–O` The host name of the computer with `O` status (original system) is output (default). If there is no computer with `O` status, the command terminates without any message.

`–B` The host name of the computer with `B` status (backup system) is output. If there is no computer with `B` status, the command terminates without any message.

`–OB` The host name of the computer with `OB` status (original and backup system) is output. If there is no computer with `OB` status, the command terminates automatically with no message.

`–A` The OBSERVE status of both computers is displayed. The output format is "*status* = *hostname*".

status can have the following values:

`REP` computer is waiting for reconnection (repair)

`ORI` original system

`BUP` backup system

`OAB` original and backup system

host1, *host2*
Host names of the computers on which OBSERVE is running.

port Communication port number as specified in the `/etc/inet/services` file under the name `obsvadm`. This parameter is optional. If it is not supplied as command line argument the communication port number will be determined automatically.

EXIT STATUS

`0` `lan_host` terminated without error.

`1` An error occurred.

SEE ALSO

`observe`(5) .

mb2_astat(1M) - Xscon(1M)

mb2_astat(1M)

NAME

`mb2_astat` - MB2 statistical evaluation

SYNOPSIS

`mb2_astat` [*statfile*]

DESCRIPTION

With the `mb2_astat` command you can evaluate the MB2 statistical data that was incorporated using the MB2 diagnostic command `mb2stat(1M)`.

statfile is a file containing the "raw data" that was obtained using the `mb2stat` command. If this is not available the "raw data" is expected from `stdin`.

The data evaluated using the command `mb2_astat` is output to `stdout`.

FILES

`/opt/diag.d/bin/mb2_astat`
`/usr/include/sys/io/mb2.h`

SEE ALSO

`mb2_atrc(1M)`, `mb2stat(1M)`, `mb2trc(1M)`.

Fujitsu Siemens Computers Add-on Reference Manual

mb2_atrc(1M)

NAME

`mb2_atrc` - MB2 trace evaluation

SYNOPSIS

`mb2_atrc` [`–g`] [*tracefile*]

DESCRIPTION

With the `mb2_atrc` command you can evaluate the MB2 trace data that was incorporated using the MB2 diagnostic command `mb2trc`(1M) .

OPTIONS

`–g` With this option an extended evaluation is performed. The DMA, Command, and Reply data is searched for the DPTG-HDLC data. If this data is found, it is evaluated and represented in the same format that is usual for DPTG-HDLC trace evaluation.

trace-file

A file containing the "raw data" that was obtained using the command `mb2trc`(1M) . If this is not available the "raw data" is expected from `stdin`.

The data evaluated using the command `mb2_atrc` is output to `stdout`.

FILES

`/opt/diag.d/bin/mb2_atrc`

`/usr/include/sys/io/mb2.h`

SEE ALSO

`mb2_astat`(1M) , `mb2stat`(1M) , `mb2trc`(1M) .

Fujitsu Siemens Computers Add-on Reference Manual

mb2stat(1M)

NAME

`mb2stat -` MB2 statistics

SYNOPSIS

`mb2stat` [–S *mb2id*] [–s *mb2id*] [–t] [–b *statbuffer*] [–r *rtime*] [–o *outfile*]

DESCRIPTION

`mb2stat` is an MB2 statistics command which you can use to incorporate the individual statistics events during communication between the controllers that are connected to the MB2 bus and the UNIX host. The individual events are defined using the `statistic\f data structure inmb2.h`.

OPTIONS

–S *mb2id*
: This option is used to switch on the MB2 statistics. An MBP is uniquely identified using the `mb2id` identifier.

–s *mb2id*
: This option is used to deactivate the global statistics. An MBP is uniquely identified using the `mb2id` identifier.

–t
: This option is used to activate the time measurement.

–b *statbuffer*
: The size of the statistics buffer.

–r *rtime*
: The time in seconds during which the statistical data is gathered. When this time has has elapsed the gathering of the statistical data is terminated, whereby "rtime = 0" signals infinite waiting.

–o *outfile*
: Output file for the statistical data. If this is not available, the statistical data is output using `stdout`.

Default values: `mb2stat -S 0 -b 380 -r 30`

FILES

`/opt/diag.d/bin/mb2stat`
`/usr/include/sys/io/mb2.h`
`/etc/default/mb2diag`

SEE ALSO

`mb2_astat`(1M), `mb2_atrc`(1M), `mb2trc`(1M).

Fujitsu Siemens Computers Add-on Reference Manual

mb2trc(1M)

NAME

mb2trc - MB2 trace

SYNOPSIS

mb2trc [–G *mb2id*] [–g *mb2id*] [–l*trclev*] [–v *dsrc*] [–n *ctrlno*] [–s *ctrltyp*] [–p *portid*] [–b *trcbuf*] [–r *rtime*] [–o *outfile*]

DESCRIPTION

mb2trc is an MB2 diagnostic command you can use to incorporate the data stream between the controllers connected to the MB2 and the UNIX host.

OPTIONS

–G *mb2id*
: This option is used to activate the global trace. An MBP is uniquely identified using the mb2id identifier.

–g *mb2id*
: This option is used to switch off the global trace. An MBP is uniquely identified using the mb2id identifier.

–l *trclev*
: This option is used to define the data segments that are logged, whereby the trace level (*trclev* = fourpart hexadecimal number) is:

  0001 SPA MI message from UNIX to MBP.

  0002 Command partition data structure: mb2partition.

  0004 Command buffer + transaction descriptor data structure: mb2iocmd + mb2transdesc

  0008 DMA data out.

  0100 SPA MI message from MBP to UNIX.

  0200 Reply partition data structure: mb2partition.

  0400 Reply buffer + transaction descriptor data structure: mb2iorpl + mb2transdesc.

  0800 DMA data in.

–v *dsrc*
: This option is used to define the trace data flow.

  0 Trace of the I/O traffic over an MBP (all controllers).

  1 MB2 port-specific traces.

  2 Traces of a logical controller.

–n *ctrlno*
: You can check a logical controller number using the command showconf(8) . This can be found in the log: column. If you select the option –v 0, the option –n *ctrlno* is not relevant.

–s *ctrltyp*
: You can check the type of the logical controller using the command showconf(8) . This can be found in the ctype: column. If you select the option –v 0, the option –s *ctrltyp* is not relevant.

–p *portid*
: If you select the options –v 0 or –v 2, the option –p*portid* is not relevant.

–b *trcbuf*
: The size of the trace buffer.

–r *rtime*

The time in seconds during which the statistical data is gathered. When this time has elapsed the gathering of the statistical data is terminated, whereby "rtime = 0" signals infinite waiting.

–o *outfile*
Output file for the statistical data. If this is not available, the statistical data is output using `stdout`.

Default values: `mb2trc -G 0 -l 0f0f -v 0 -b 16384 -r 20`

FILES

`/opt/diag.d/bin/mb2trc`
`/usr/include/sys/io/mb2.h`
`/etc/default/mb2diag`

SEE ALSO

`mb2_astat`(1M), `mb2_atrc`(1M), `mb2stat`(1M), `showconf`(8).

Fujitsu Siemens Computers Add-on Reference Manual

mount(1M-hs)

NAME

`mount(hs)` - mount hs file systems

SYNOPSIS

`mount` [`-F hs`] [*generic_options*] [`-r`] [`-o`*specific_options*] {*special* | *mount_point*}

DESCRIPTION

`mount` attaches a `hs` file system to the file system hierarchy at the pathname location *mount_point*, which must already exist. If *mount_point* has any contents prior to the `mount` operation, these remain hidden until the file system is once again unmounted.

OPTIONS

`-F hs`
: Specifies the `hs`-FSType.

`-r`
: Mount the file system read-only (redundant). This is the system default.

`-o`
: Specify `hs` file system specific options. If invalid options are specified, a warning message is printed and the invalid options are ignored. The following options are available:

  `ro`
  : Mount the file system read-only (redundant).

  `nosuid|suid`
  : By default the file system is mounted with setuid execution allowed. Specifying `nosuid` overrides the default and causes the file system to be mounted with setuid execution disallowed.

  `remount`
  : Used in conjunction with `rw`. A file system mounted read-only can be remounted read-write. Fails if the file system is not currently mounted or if the file system is mounted `rw`.

  `dos`
  : This option is used for turning on filename mapping for DOS applications on exported file systems.

  `joliet`
  : This option is used for turning on filename mapping for the joliet extensions

  `rdev=`*devicename*
  : For locking and unlocking of the drive the raw device of the cd-drive is needed. This can be achieved by using `rdev=`*devicename*. If this option is missing and the block device name is an absolute path name the character device name is build using the pathname of the block device by putting an "r" in front of the block device (e.g. `/dev/ios0/sdisk006s0` leads to `/dev/ios0/rsdisk006s0`).

  {`uid|u`}=*name*
  : The semantics of the option can be taken from the description of the `cdmntsuppl`(1M) command.

  {`gid|g`}=*name*
  : The semantics of the option can be taken from the description of the `cdmntsuppl`(1M) command.

  {`dperm|D`}=*mode*
  : The semantics of the option can be taken from the description of the `cdmntsuppl`(1M) command.

  {`fperm|F`}=*mode*
  : The semantics of the option can be taken from the description of the `cdmntsuppl`(1M) command.

  {`uidmap|U`}=*file*

The semantics of the option can be taken from the description of the cdmntsuppl(1M) command.

{gidmap|G}=*file*
The semantics of the option can be taken from the description of the cdmntsuppl(1M) command.

I={*file*|–}
Input is read from *file* or from standard input. The semantics of the option can be taken from the description of the cdmntsuppl(1M) command.

nmconv=*value*
The value c, l, m, lm or ml can be specified. The semantics of the option can be taken from the description of the cdmntsuppl(1M) command.

dsearch=*value*
The value x or s can be specified. The semantics of the option can be taken from the description of the cdmntsuppl(1M) command.

devmap=*mapfile*
The semantics of the option can be taken from the description of the cddevsuppl(1M) command.

norrip
You can use norrip to suppress evaluation of the RRIP data on the CD-ROM.

NOTES

If the directory on which a file system is to be mounted is a symbolic link, the file system is mounted on the directory to which the symbolic link refers, rather than on top of the symbolic link itself.

FILES

/etc/mnttab
table of mounted file systems

SEE ALSO

generic mount(1M), mkdir(2), mount(2), umount(2), open(2), mnttab(4), hs(7).

Fujitsu Siemens Computers Add-on Reference Manual

muxadm(1M)

NAME

`muxadm -` administration of the DPTG2 monitor

SYNOPSIS

`/usr/sbin/muxadm –F –a` *path* `–d` *device* **[** `–h` *host* **]** **[** `–l` *label* **]** **[** `–s` *service* **]**

`/usr/sbin/muxadm –I` **[** `–t` *level* **]** **[** `–f` *file* **]** **[** `–v` *version* **]** **[** `–r` *retry* **]** **[** `–x` **]** *pmtag*

`/usr/sbin/muxadm –T` **[** `–t` *level* **]** *pmtag*

`/usr/sbin/muxadm –S` **[** `–i` *svrtag* **]** *pmtag*

`/usr/sbin/muxadm –R` *pmtag*

`/usr/sbin/muxadm –V`

`/usr/sbin/muxadm –?`

DESCRIPTION

`muxadm` facilitates the administration and configuration of DPTG2 monitors. `muxadm` has, for example, the `–R` option, with which an entry specific to DPTG2 monitor is output to the `_pmtab` file in the correct format.

In addition, a monitor incarnation can be generated with the `-I` option; this type of incarnation can be removed using the `–R` option.

The `–V` option provides `stdout` with the version number of the `_pmtab`-specific format.

The monitor status and the status of the connections it is monitoring are displayed using the `–S` option.

Finally, trace mode can be selected and stopped using the `–T` option.

Each option has a series of arguments, which are explained in detail below.

Option –F (Format)

With this option, a DPTG2 monitor-specific part of a `_pmtab` entry is returned to `stdout` without line feed. The format of this entry corresponds to the version number provided by the `–V` option (see below). Other options and arguments include:

`–a` *path*

This is the path name of the administrator channel via which the multiplexer is to be connected to a line. Specification of this name can be relative or absolute. If the path name is specified relatively, the directory `/dev/dptg` is sought.

Warning:

Specification of the path name is mandatory (otherwise diagnostic error, No. 5, `E_NOADM` occurs).

`–d` *device*

This option is used to select the physical connection of the device *device* which is to be monitored.

If it is a *direct connection*, the name given here is that of the asynchronous tty port. This name can be specified relative to `/dev/term` or absolutely.

If it is a *remote* connection via TCP/IP, the *absolute* port number is specified.

Warning:

Specification of *device* is mandatory (otherwise diagnostic error No. 6, `E_NODEV` occurs).

`–h` *host*

With this option, the host on which the multiplexer connection is to be made is specified (PC or a terminal server). If it is a *direct connection*, the name of the host is `localhost`, or it can be omitted completely. This field remains empty by default. Specification of the *remote* host is made either symbolically (corresponding to an `/etc/hosts` entry), or in Internet (dot) notation.

–l *label*

With this option, *label* denotes a tag in /etc/ttydefs, after which the monitor must set the physical line parameters. The file is only evaluated in the case of a *direct connection*. If this option is not used, the tag 9600 is used.

Modification of the physical line parameters via a DPTG2 monitor can only be carried out by changing this *label* tag.

–s *service*

This option determines whether the DPTG2 monitor must carry out a load service before the DPTG2 protocol can begin. This load service is specified in more detail with the argument *service*. If *service* has the value N, no load service is carried out (e.g. in the case of the connection of a PC emulation). If the value is Y, the load service is carried out. Here, the DPTG2 terminal determines which workstation program is loaded (setting in the terminal's SETUP menu). As a third option, the path name of the workstation program can be specified. If only a relative path name was selected, the disk program relative to /opt/xb is sought. The default value is Y.

Option –I (Install)

This option controls the way in which sac(1M) creates a DPTG2 monitor incarnation (muxmon) with the name *pmtag*. An entry is made in the file /etc/saf/_sactab. Other options and arguments include:

–t *level*

With this option, trace mode is set when the muxmon monitor is initialized. The argument *level* specifies the trace level. Trace levels are:

0 No output (trace mode switched off).

1 Output of error messages only.

2 Output of trace information and the contents of data structures.

The default value is 0 (trace mode switched off).

–f *file*

The DPTG2 monitor uses /var/saf/*pmtag*/trace as a trace file by default. You can write the trace output to another file with this option by creating a DPTG2 monitor incarnation. The argument *file* must specify the absolute path name of this file.

–v *version*

The selectable version here is the version tag for the formats of /etc/saf/*pmtag*/_pmtab entries. If this option is not used, the format is identified by the version which can also be obtained using the muxadm –V command.

–x This option specifies that the sac(1M) process of this DPTG2 monitor incarnation is always started as disabled. If this option is not used, the monitor is started as enabled.

–r *retry*

This option starts a muxmon, whose retry counter is set to the value *retry*. For information on the functions of the retry counter, see –r for muxmon(1M) .

Option –T (Trace)

This option is used to switch on and off trace mode. This is done by means of signals sent to the monitor with the specified *pmtag*. Another option is:

–t *level*

The trace mode of the muxmon monitor is set with this option. The argument *level* specifies the trace level. The trace levels are:

0 No output (trace mode switched off).

1 Output of error messages only.

2 Output of trace information and the contents of data structures.

The default value is 0 (trace mode switched off).

Option –S (Status)

With this option, the DPTG2 monitor for *pmtag* is requested to write it's own status and the status of the connections it is monitoring to the file `/var/saf/`*pmtag*`/state`. The contents of this file are output to `stdout`. Another option is:

`–i` *svrtag*

When this option is used, the entire status file is not output. Instead, the file is searched for the tag *svrtag*, and the appropriate line is output. The search is carried out with `fgrep(1)`, so that in the event of an error, the `fgrep` diagnostics texts can be viewed.

Option –V (Version)

This option outputs the version tag of the DPTG2 monitor-specific `_pmtab` format to `stdout` (without line feed). This command can be used when generating a DPTG2 monitor incarnation (`sacadm –a`).

Option –R (Remove)

With this option, the `muxmon(1M)` incarnation is removed from the SAF system (Service Access Facility). The *pmtag* specified is removed from the file `/etc/saf/_sactab`. The command used is `sacadm(1M)`.

Option –? (Help)

The command syntax is displayed on `stderr`.

WARNING

Trace mode should only be switched on for a short period of time, so that the trace file does not become too large.

DIAGNOSTICS

If no error occurs, `muxadm` terminates with the return value `0`. In the event of an error, the return values have the following meanings:

`1` = Illegal option was used (`E_ILLOPT`)

`2` = Command was called without any parameters (`E_NOPARAM`)

`3` = No command options were specified (`E_NOCMD`)

`4` = Command requires a *pmtag* (`E_NOTAG`)

`5` = Command requires specification of an administrator channel (`E_NOADM`)

`6` = Command requires specification of a device (port) name (`E_NODEV`)

`7` = Command can only be executed by superuser (`E_PERM`)

`8` = The service tag specified does not exist (`–i` option) (`E_SVRTAG`)

`9` = The monitor tag specified does not exist (`E_NOMON`)

Entries under Logging 3.0 are made under the component number 36 and the error number 7.

FILES

`/etc/saf/_sactab`

`/etc/saf/`*pmtag*`/_pmtab`

`/var/saf/`*pmtag*`/state`

`/var/saf/`*pmtag*`/trace`

SEE ALSO

`muxmon(1M)`, `sac(1M)`, `sacadm(1M)`, `dptg2(7)`.

Fujitsu Siemens Computers Add-on Reference Manual

muxmon(1M)

NAME

muxmon - DPTG2 monitor

SYNOPSIS

/usr/lib/saf/muxmon [–d *level*] [–f *file*] [–t *sec*] [–r *retry*]

/usr/lib/saf/muxmon –?

DESCRIPTION

muxmon is a port monitor based on TTY-STREAMS, as prescribed by System V Release 4 through SAF (Service Access Facility). muxmon's job is to monitor links to the terminals, which are operated using the DPTG2 protocol. muxmon is designed such that it usually only runs under the control of the SAC (Service Access Controller) [see sac(1M)]. muxmon can be operated using the sacadm(1M) and pmadm(1M) commands. DPTG2-specific commands and formats can be executed using the muxadm(1M) command.

The basic functions of muxmon are:

1) General control of monitor processes

This function controls the actual selection of the monitor. Under SAF, the monitor has 2 internal statuses: ENABLED and DISABLED. The monitor is started in either of the two statuses from SAC. The status taken when the monitor is started defines the _sactab entry. If DISABLED, none of the ports is monitored. If ENABLED, all terminals listed in the _pmtab file are monitored. The monitor is terminated using a SIGTERM(15) signal.

2) Monitoring of links

Each entry in the monitor _pmtab file describes a link to a DPTG2 terminal. In accordance with this entry, the monitor opens the link to the terminal and monitors its status. The multiplexer is activated when the terminal is switched on (I_LINK) [see termio(7)]. The multiplexer is disconnected when the terminal is switched off (I_UNLINK) [see termio(7)]. When the monitor status is DISABLED or the _pmtab entry reads x=disable, the multiplexer administrator channel is closed.

3) Execution of tasks

A load service can be specified in the terminal _pmtab entry. If required (e.g. when operating a 9766 terminal), the program to be loaded can be indicated using the name of a workstation program (e.g. xb4D). If N is selected, the load service is switched off. If Y is selected, the terminal can specify with which program it is to to be loaded. Loading is always completed prior to activation of the multiplexer (I_LINK). Until loading is completed, an open system call is blocked on the user channels.

OPTIONS

If muxmon is invoked and options have not been specified, the following default values apply:

–d *level*

This option sets the trace level to a value of *level*. If the value is greater than 2, it is reduced to 2. The default value is 0.

–f *file*

This option can be used to select the *file* trace file instead of the /var/saf/*pmtag*/_trace default file.

–t *sec*

This option sets the timeout for repetitions to *sec*. If *sec* is less than 60 seconds, it is increased to 60. The default value is 90 seconds.

–r *retry*

In all error situations, the links are reset to the corresponding connecting terminals (service ID) for a timeout [see option –t]. When the timeout has expired, an attempt is made to reestablish the link. In the event of a further error, the entire process is repeated. The process is repeated MAXRETRY times. If

all attempts fail, the service ID is `DISABLED`. The default for `MAXRETRY` is 12. This option can be used to change the default value when the monitor is started.

The *retry* argument can have any integer value. However, values lower than 5 are increased to 5. In the case of negative values, the absolute value applies. If –1 is selected, continuous attempts are made to reestablish the link. Only the first and last attempts are recorded in the `/var/saf/`*pmtag*`/log` logging interface. Thus, in the case of a –1 value, only the first attempt is recorded.

`–?` Command syntax is displayed on `stderr`.

DIAGNOSTICS/MAINTENANCE

For diagnostic purposes, the monitor keeps a private logging file called `log` in the `/var/saf/`*pmtag* directory. This file can only be edited by the monitor. This log file records the SAC commands, status changes and error situations.

As a Reliant UNIX system component, the DPTG2 monitor also writes all of these log records to the Reliant UNIX logbook. They are filed under component number 36 and error number 6. As the `muxmon` files all of its messages under this error number, the repetition counter in the logging system for the component group should generally be switched off.

To facilitate the search for errors, it is possible to operate the monitor in a trace (debugging) mode. To do this, a file is specified under option `–f` which contains the trace information. If the `–f` is not used, the monitor writes this trace information to `trace` in the `/var/saf/`*pmtag* directory. The trace mode is activated by the `–d` option when the monitor is started. It can also be activated dynamically by signals later at runtime. Trace mode can be operated at 3 levels:

`0` No trace output.

`1` Output only in the case of error.

`2` Full trace output, which can be used to trace the operational course of the monitor.

These traces are intended only for diagnostic use by developers. Trace mode should only be activated when required.

The `SIGUSR1`(16) signal increases the trace level by one. `SIGUSR2`(17) sets the trace level to 0. Trace level can also be set using the `–d` option at monitor start. There is no administration of the trace file. This means that trace mode should not be activated under normal operating conditions (level 0), otherwise the trace file could overflow.

The status of the links to be monitored are written to a status file called *state* by the `SIGHUP`(1) signal, which is located in the `/var/saf/`*pmtag* directory. Previous status entries in the file are overwritten.

The *state* file has the following format:

`DPTG-monitor '`*pmtag*`'` (pid=*number*) state: *pm_state*

whereby *pmtag* denotes the name of the monitor incarnation. The PID *number* is the current process ID, *pm_state* can accept both `PM_DISABLED` and `PM_ENABLED`.

The entry for each link to be monitored occupies one line. There are different formats according to the type of entry.

A *remote* link looks like this:

no`.` *svrtag*`: admin` *path*`,` [*state*] `host '`*name*`' port` *port_no* `=>` *status*\fH

and a *direct link*:

no`.` *svrtag*`: admin` *path*`, local device '`*name*`' =>` *status*\fH

The format fields are as follows:

no The serial number of an entry.

svrtag
The symbolical tag name for the link (service ID).

path Path name of administrator channel to multiplexer.

state

For remote links, the availability of the host is tested from time to time. The result of the most recent check is displayed here.

name For direct links, the path name of the serial terminal connection is displayed here. For remote links, the host name is displayed here as entered in the *_pmtab* file.

port_no
For remote links, the port number for the remote host is displayed here.

status
This field shows the status of the terminal connection as read by the monitor. The options are:

`SUSPENDED`
An error occurred in establishing the link. The reason for the suspension is always indicated after the keyword `reason:`

`DISABLED`
This status is registered when an attempt to reestablish a connection `MAXRETRY` has failed once. The reason for the most recent failure is indicated after the keyword `reason:`

`ACTIVATED`
This indicates that a connection establishment sequence has begun. The administrator channel is already open, and the monitor is waiting for confirmation of an "Open" or acceptance of a "Connect".

`CONNECTED`
This status indicates that a link has been established with the terminal.

`LOADING`
Contact has been established with the terminal and loading is in process.

`LINKED`
The terminal is available and loaded. It is ready for DPTG2 operation.

WARNING

Trace mode should only be switched on for a short period of time, so that the trace file does not overflow.

DIAGNOSTICS

If no error occurs, `muxmon` terminates with a return value of 0. In the event of an error, the return values have the following meanings.

`1` = An illegal option was used (`E_ILLOPT`)
`2` = The `_pid` file could not be opened (`E_PID`)
`3` = The `log` file could not be opened (`E_LOG`)
`4` = No exclusive access to the `_pid` file (`E_PLOCK`)
`5` = No exclusive access to the `log` file (`E_LLOCK`)
`6` = An internal pipe could not be opened (`E_PIPE`)
`7` = Illegal monitor start status (`E_ISTATE`)
`8` = `_pmpipe` file not available (`E_PIPE`)
`9` = `_sacpipe` file not available (`E_SACPIPE`)
`10` = Error reading `_pmpipe` file (`E_ERROR`)
`11` = Error reading `_sacpipe` file (`E_SACMSG`)
`12` = Wrong format version in `_pmtab` file (`E_VERSION`)

FILES

`/etc/saf/_sactab`
`/etc/saf/_sacpipe`
`/etc/saf/`*`pmtag`*`/_pmtab`

/etc/saf/*pmtag*/_pmpipe
/var/saf/*pmtag*/state
/var/saf/*pmtag*/log
/var/saf/*pmtag*/trace

SEE ALSO

muxadm(1M), sac(1M), sacadm(1M), dptg2(7), termio(7).

Fujitsu Siemens Computers Add-on Reference Manual

node_add(1M)
Cluster Operating System

NAME

node_add - add a node to a cluster

SYNOPSIS

node_add [–C *clos_name*] –n [*tag_name*,]*inet* [–v *string*] [–n [*tag_name*,]*inet* [–v *string*]] ...

node_add –?

DESCRIPTION

This command can be used to add one or more nodes to the *clos_name* cluster. At least one *inet* argument must be specified with the –n option. The *string* arguments for the –v option are assigned to the node identifiers of the –n option on the basis of the call sequence. If the –v option was used for a node, it must be used for all nodes. Otherwise, the command is rejected.

A distinction is made between the name of the node in the network (Internet name) and the name of the node in the cluster (tag name) [see clos(7)]. The network names of the nodes are defined in the standard files on the operating system and can be used in various forms for node administration: official network name, alias or Internet dot notation. Each of the three forms can be used in the *inet* argument of the –n option. The tag names are defined by the cluster administrator. The get_uname(1M) is used to check, where possible, whether the selected tag name corresponds to the system name [see –n option of uname(1)]. This only happens if communication is possible with the node on the basis of the network name. Otherwise, a warning is issued to standard error output.

The node_add command guarantees the consistency of the COD archive. The consistency is checked initially, and if inconsistencies are discovered, the command aborts with errors. If the calling node itself is added to the cluster, it is assumed that no node was previously configured in the cluster and thus no COD archive also existed. In this case, the command produces an archive initially.

Since the database of the Closmon monitor [see closmon(1M)] must be updated for the node_add command, this update must be made known both to the local monitor and to the monitors of the other nodes in the cluster. For this reason, the cluster object database [CLODB, see clodb(7)] is distributed consistently to all nodes in the cluster at the end of this command [see cod_sync(1M)]. If a node is not integrated, for example, when a node_add command is executed, a recovery script is generated for this node; as soon as the node is integrated back into the cluster, it also receives the newly created COD archive.

If the node(s) being added is (are) not accessible via the *inet* address when this command is executed, no checks can be performed. These checks are then made up the first time such nodes can be accessed in the network. If the check is successful, the node is then integrated. If the check fails, the node is removed fully from the cluster. The subsequent addition is flagged in the log file of the Closmon monitor.

OPTIONS

–C *clos_name*

With this option, the name of the cluster is selected in the *clos_name* argument. If this option fails, the name from the $CLOSNAME environment variable is used or, if this is not set, the name is taken from the /var/clos/.CLOSdefault file.

–n [*tag_name*,]*inet*

Choosing the name allocation.

The –n option is used to establish the allocation of tag names to network names. The argument for this option is a pair of names separated by a comma. The first name, *tag_name*, is the desired tag name of the node while the second name, *inet*, is the network name to be assigned. The format of the network name can be chosen freely (alias, full name or Internet dot notation).

The command establishes the associated node name for each *inet* argument [see uname(1)] and

compares it, if specified, with the *tag_name* argument of the –n option, if it exists. If the two names are different, the command aborts with the return value 9. If the node to be added is not accessible when the command is called, no check is performed and both arguments are accepted after a warning has been issued. If no *tag_name* argument has been used, the node name is used as the tag name.

This option must be used at least once but can also be used several times in a command. In this case, several nodes are added simultaneously. Passwords are then assigned to the nodes based on the call sequence.

–v *string*
The string transferred with this option is sent as an encrypted password to the new node being added and a check is made to establish whether this is the password for the superuser (root) on the new node. If this option is not used, the password is established interactively. The command aborts if the password is incorrect.

–? Usage of this command.

DIAGNOSTICS

The node_add command terminates with the return value 0 if no errors occur. The nodes in question are added to the cluster or can also be added subsequently when they have booted. If an error is detected, on the other hand, a message is issued and the return value is greater than 0. No node is added in the event of errors!

Error messages are sent to standard error output along with the following return values:

1 "bad n-option argument". Incorrect call: Command option –n contains an incorrect argument.

2 "you need to be superuser". Incorrect call: The command may only be performed by a superuser.

3 "Can't assign CLOS name". Incorrect call: The name of the cluster cannot be established.

4 "CLOSMON '*CLOSNAME*' not running". Situation error: The local node is not integrated in the specified cluster.

5 "net name *name* multiple defined". Situation error: The network name is specified more than once.

6 "System Access Facility failed!". Situation error: The pmadm(1M) command failed.

7 "tag *tag_name* multiple defined". Situation error: The tag name is specified more than once.

8 "no such host available". Situation error: When using the Add command, the specified system could not be found over the network.

9 "hostname '*host*' of node '*node*' doesn't match". Situation error: It was discovered with the Add command that the tag name was already assigned as a system name for another node.

10 "Can't resolve ipname". Situation error: The IP address of the node could not be established.

11 "permission denied". Call error: The password string transmitted for the command was incorrect.

12 "v-option needed". Situation error: One of the nodes to be added is not in the network. A subsequent password check can only be performed if the –v option was used.

13 "v-option count doesn't match arg count". Call error: The number of –n options does not match the number of –v options.

14 "COD archive is not consistent". Situation error: If the COD archive is not consistent, no node is added.

15 "COD archive busy". Situation error: A command is currently active that precludes the possibility of modifying the COD archive.

16 "Can't sync the COD archive". Situation error: The consistent distribution of the COD archive failed [see cod_sync(1M)].

17 "Interrupted!". Situation error: The command aborted with a signal.

FILES

`/var/clos/`*clos_name*`/COD`

SEE ALSO

`uname`(1), `closmon`(1M), `cod_sync`(1M), `get_uname`(1M), `pmadm`(1M), `clos`(7).

Fujitsu Siemens Computers Add-on Reference Manual

node_adm(1M)
Cluster Operating System

NAME

node_adm - administration of cluster nodes

SYNOPSIS

node_adm [–C *clos_name*] –A –n [*tag_name*,]*inet* [–v *string*] [–n [*tag_name*,]*inet* [–v *string*]] ...

node_adm [–C *clos_name*] –D [–C *clos_name*] [*tag_name* ...]

node_adm [–C *clos_name*] –E [*tag_name* ...]

node_adm [–C *clos_name*] –K [*tag_name* ...]

node_adm [–C *clos_name*] –L [–l] [–t *tag_name*] ...

node_adm [–C *clos_name*] –Q

node_adm –R [*tag_name* ...]

node_adm –?

DESCRIPTION

The node_adm command is used to manage the nodes in a cluster. This can involve adding or removing nodes in the cluster group. In addition, the status of the nodes can be changed and queried.

A distinction is made in node management between the name of the node in the network (Internet name, node name) and the name of the node in the cluster (tag name) [see clos(7)].

The network names of the nodes are defined in the usual operating system files and can be used in various forms for node management: official network name, alias or Internet dot notation. Each of the three forms can be used for node_adm in the *inet* argument. The tag names are specified by the cluster administrator. Tag names are assigned to network names throughout the cluster using the Add command. If no tag name is specified, the node name of the system belonging to *inet* is used as the tag name.

Since some commands (Add, Remove) require that the Closmon monitor database [see closmon(1M)] be updated, this update must be indicated to both the local monitor and the monitors of the remaining nodes in the cluster. The /etc/saf/*clos_name*/_pmtab file is a component of the Cluster Object Database [see clodb(7)]. When the database is modified, the node_adm commands ensure that this file is distributed to all accessible nodes in the cluster. However, a cod_sync(1M) command is not performed which means that it is not possible to guarantee the consistency of the COD archive. If a node is not integrated for example when executing a node_adm command, which results in the COD archive being modified, then the modified /etc/saf/*clos_name*/_pmtab file is not copied to the node either. The file will only be copied to the node after cod_sync(1M) is called. Nonetheless, a copy of the modified /etc/saf/*clos_name*/_pmtab file is distributed as far as possible which results in the COD archive being inconsistent on all accessible nodes.

COMMAND OPTIONS

–A (Add) – Adding nodes

This command allows you to add one or more nodes to the cluster group. At least one *inet* argument must be specified with the –n option.

A node can only be added if the password entered by the superuser of the new node is correct. The –v option can be used to transfer the encrypted password. If this option is not used, the password is queried interactively. The password must be transferred in the same way for all new nodes (either using the –v option or interactively).

The /etc/saf/*clos_name*/_pmtab file is modified. The local Closmon monitor [see closmon(1M)] then has to interpret this modified file again before it is distributed to all accessible nodes specified within the file. After the file has been distributed, all remote nodes [see clos(7)] are also required to interpret their database once again.

Consistency of the COD archive is not guaranteed.

The command determines the accompanying node [see uname(1)] for every *inet* argument of the –n option and compares it, if specified, with the *tag_name* argument of the –n option, if available. If the two names differ, the command is aborted with the return value 17. If the node to be added is not accessible when the command is called, the check is not performed and both arguments are accepted after a warning is output. If no *tag_name* argument was used, the node name is used as the tag name.

–D (Disable) – Disabling communication

This command changes the status of the selected nodes to DISABLED. In this mode, no communication is mapped from the Closmon monitor to the remote node monitors. The applications that communicate over node channels receive EOF messages and the STREAMs for the node that is DISABLED are locked. Despite this, the CLOS commands still continue to function for this node via the cluster service [see clserver(1M)].

The command receives the node tag names to be addressed in the following arguments. If no tag name is specified, all nodes are selected.

–E (Enable) – Enabling communication

This command changes the status of the selected nodes to INTEGRATED. If communication was previously interrupted, it is now enabled. If the node was already INTEGRATED, then this command has no effect. If, however, the status of the node was either NOT_AVAIL or DISABLED, it is switched.

The command receives the node tag names to be addressed in the following arguments. If no tag name is specified, all nodes are selected.

–K (Kill) – Ending node communication

This command terminates the Closmon monitors on the selected nodes. This is achieved by executing the sacadm –k –p *clos_name* command on each node. The monitors close their communications channels and signal to the remaining nodes that they are no longer available (NOT_AVAIL status).

The command receives the node tag names to be addressed in the following arguments. If no tag name is specified, all nodes are selected.

–L (List) – Listing the various types of node status

The configured nodes in the cluster are displayed in accordance with the /etc/saf/*clos_name*/_pmtab file. If no tag name is selected, all configured nodes in the cluster are listed. Otherwise, only the selected nodes are displayed.

The user can choose between a short and a long format (–l option). The status is displayed from the CLOS point of view for every node. The various possible states are as follows:

NOT_AVAIL
: the node is not accessible

INTEGRATED
: the node is integrated in the cluster

DISABLED
: the node is not released to engage in communication

–Q (Query) – Querying

This command allows you to query whether the local node is configured in the specified cluster. If so, the response is sent to standard output and the return value is 0. If the node is not configured, the return value is 11 and is accompanied by the error message documented below.

–R (Remove) – Removing nodes

This command removes one or more nodes from the cluster group.

The entry for the node to be removed is deleted in the /etc/saf/*clos_name*/_pmtab database on all accessible nodes in the CLOS name space (with the exception of the local node). This results in the accompanying *tag_name* directory also being deleted in /var/clos/*clos_name*/nodes on all of these nodes.

Following this, all *tag_name* directories in the /var/clos/*clos_name*/nodes directory and the

/etc/saf/*clos_name*/_pmtab database are deleted on the node to be removed. The Closmon monitor on the deleted node is not ended, rather it now works on an empty database.

Finally, the database entries to be deleted and the *tag_name* directories in the local node on which the command was executed are deleted.

Consistency of the COD archive is not guaranteed.

The command receives the node tag names to be deleted in the following arguments. If no tag name is specified, all nodes are selected.

OPTIONS

−C *clos_name*

The cluster name is selected in the *clos_name* argument using this option. If this option is not specified, the name from the $CLOSNAME environment variable is used or, if this is not set, the name is taken from the /var/clos/.CLOSdefault file.

−l Selecting list format.

If this option is not specified, a short format is output for every node using the List command:

tag_name status

If this option is selected, a header is first output and each node entry also receives the system name, the operating system version and the machine ID [see taginfo(1M) , −h option].

−n [*tag_name*,]*inet*

Selecting a name assignment.

The −n option determines the assignment of tag names to network names. The argument for this option is a pair of names separated by a comma. The first *tag_name* name is the desired node tag name and the second *inet* name is the network name to be assigned. The network name can have any format (alias, full name or Internet dot notation).

If the tag name is missing from the argument, this is established by node_adm using the *inet* network name, whereby the name is the same as the node name [see uname(1) , −n option] of the accompanying system.

This option is mandatory for the Add command. It can however be used more than once in a command. This will result in several nodes being added simultaneously.

−t *tag_name*

Selecting a tag name.

An individual tag name is selected for the List command using the −t option. Several names can be selected through multiple use of this option. If no tag name is selected, all tag names are displayed.

−v *string*

The string transferred with this option is sent as an encrypted password to the new node being added and a check is made to ensure that this is the password for the superuser (root) on the new node. If this option is not used, the password is established interactively. The command aborts if the password is incorrect.

−? Usage of this command.

DIAGNOSTICS

The node_adm command terminates with the return value 0 if no errors occurred. Error messages are output to standard error output with the following return values:

1 "use command option exclusively". Incorrect call: Command options are exclusive.

2 "wrong tag option with *cmd* command". Incorrect call: The −t option was used mistakenly in the specified command.

3 "wrong name pair option used". Incorrect call: The −n option is only permitted with the Add command.

4 "`bad name pair argument`". Incorrect call: A pair of names, separated by a comma, must be specified for the `-n` option.

5 "`You must select -A, -D, -E, -K, -L or -R command`". Incorrect call: A command must be selected.

6 "`you need to be superuser`". Incorrect call: Only the superuser may execute the command.

9 "`Can't assign CLOS name`". Incorrect call: The name of the cluster cannot be determined.

10 "`CLOSNAME not available! CLOSMON not running`". Situation error: The local node is not configured in the specified cluster.

11 "`tag` *tag_name* `not known in $CLOSNAME`". Situation error: The specified tag name is not known in the cluster.

12 "`net name` *name* `multiple defined`". Situation error: The network name is specified more than once.

13 "`tag` *tag_name* `multiple defined`". Situation error: The tag name is specified more than once.

14 "`CLOSNAME: system access facility failed!`". Situation error: The `pmadm`(1M) command failed.

15 "`Remove from CLOSNAME: system access facility failed!`". Situation error: The `pmadm`(1M) command failed.

16 "`no such host available`". Situation error: When using the `Add` command, the specified system could not be found over the network.

17 "`hostname '`*host*`' is assigned to nodename '`*node*`' and doesn't match`". Situation error: When using the `Add` command, it was discovered that the tag name was already assigned as a system name for another node.

18 "`No such CLOS '`*clos_name*`' configured`". Situation error: No cluster available with the specified name.

20 "`permission denied`". Call error: The password string transmitted for the command was incorrect.

21 "`v-option needed`". Situation error: One of the nodes to be added is not in the network. A subsequent password check can only be performed if the `-v` option was used.

22 "`v-option count doesn't match arg count`". Call error: The number of `-n` options does not match the number of `-v` options.

FILES

`/etc/saf/`*clos_name*`/_pmtab`

`/var/clos/.CLOSdefault`

`/var/clos/`*clos_name*`/nodes`

`/var/clos/`*clos_name*`/nodes/`*tag_name*

`/var/clos/`*clos_name*`/COD`

SEE ALSO

`closmon`(1M), `cod_sync`(1M), `pmadm`(1M), `sacadm`(1M), `taginfo`(1M), `clos`(7).

Fujitsu Siemens Computers Add-on Reference Manual

node_rm(1M)
Cluster Operating System

NAME

node_rm - remove a node from a cluster

SYNOPSIS

node_rm [−C *clos_name*] [−f] *tag_name* [*tag_name* ...]

node_rm −?

DESCRIPTION

This command can be used to remove one or more nodes from the *clos_name* cluster. At least one tag_name argument must be specified.

The node_rm command guarantees the consistency of the COD archive. The consistency is checked initially, and if inconsistencies are discovered, the command aborts with errors. Since the database of the Closmon monitor [see closmon(1M)] must be updated for the node_rm command, this update must be made known both to the local monitor and to the monitors of the other nodes in the cluster. For this reason, the cluster object database [CLODB, see clodb(7)] is distributed consistently to all nodes in the cluster at the end of this command [see cod_sync(1M)]. If a node is not integrated, for example, when a node_rm command is executed, a recovery script is generated for this node; as soon as the node is integrated back into the cluster, it also receives the newly created COD archive.

If the node(s) being removed is (are) not accessible when this command is executed, this node's membership of the cluster cannot be terminated. A suitable warning is issued. The node is removed for the remaining cluster and the cluster continues to function. The node tries to reintegrate itself back into the cluster at a later time, but is rejected by the other nodes. In the case of this node, all other nodes have the status "NOT_AVAIL".

OPTIONS

−C *clos_name*
: With this option, the name of the cluster is selected in the *clos_name* argument. If this option fails, the name from the $CLOSNAME environment variable is used or, if this is not set, the name is taken from the /var/clos/.CLOSdefault file.

−f
: The command is executed even if the COD archive is inconsistent. The invoking of the consistency check for the cluster object database [see clodb(7)] is suppressed and does not cause an abort if inconsistent.

−?
: Usage of this command.

DIAGNOSTICS

The node_rm command terminates with the return value 0 if no errors occur. The nodes in question are removed from the cluster. The COD archive is consistent. If an error is detected, on the other hand, a message is issued and the return value is greater than 0. No node is removed in the event of errors!

Error messages are sent to standard error output along with the following return values:

1
: "illegal usage". Incorrect call: Command contains incorrect argument.

2
: "you need to be superuser". Incorrect call: The command may only be performed by a superuser.

3
: "Can't assign CLOS name". Incorrect call: The cluster name cannot be established.

4
: "CLOSMON '*CLOSNAME*' not running". Situation error: The local node is not integrated in the specified cluster.

5
: "tag not configured". Situation error: The specified tag name is not configured in the relevant

cluster.

6 "`No tag configured`". Situation error: No tag name is configured in the relevant cluster.

7 "`System Access Facility failed!`". Situation error: The `pmadm`(1M) command failed.

14 "`COD archive is not consistent`". Situation error: The `COD` archive is not properly installed or is inconsistent.

15 "`COD archive busy`". Situation error: A command is currently active that precludes the possibility of modifying the `COD` archive.

16 "`Can't sync the COD archive`". Situation error: The consistent distribution of the `COD` archive failed [see `cod_sync`(1M)].

FILES

`/var/clos/`*clos_name*`/COD`

SEE ALSO

`closmon`(1M), `cod_sync`(1M), `clodb`(7), `clos`(7).

Fujitsu Siemens Computers Add-on Reference Manual

nodenum(1M)

NAME

`nodenum` - return a node number for the local node

SYNOPSIS

`/sbin/nodenum`

DESCRIPTION

The `nodenum` command returns a node number for the node on which it is executed. If the node number has not yet been initialized, the number "0" is returned. The `nodenum` command always terminates with exit status 0.

SEE ALSO

`nodeinfo`(1), `icfmap`(1M).

Fujitsu Siemens Computers Add-on Reference Manual

obsallow(1M)

NAME

`obsallow -` allow OBSERVE start

SYNOPSIS

`obsallow`

DESCRIPTION

The command `obsallow` allows OBSERVE to be automatically started after the next reboot, thus enabling it to monitor the two node cluster.

EXIT STATUS

`0`	`obsallow` terminated without error.
`1`	Invalid parameter or syntax error.
`7`	Command aborted – couldn't create file `/opt/observe/.internal/.obsstart`.
`9`	Initialization error in message component.
`255`	Command not allowed in this operation mode.

SEE ALSO

`obsforbid`(1M), `observe`(5).

Fujitsu Siemens Computers Add-on Reference Manual

obschange(1M)

NAME

obschange - exchange original and backup computer functions

SYNOPSIS

obschange

DESCRIPTION

The computer functions in a two node cluster can be swapped using the obschange command. The original system becomes the backup system and the backup system becomes the original system.

The command is only permissible if OBSERVE is started on both machines and one computer is defined as the original system (computer status O) and the other as the backup system (computer status B).

The obschange function can be prohibited by setting the OBSCHANGE environment variable in the /etc/default/observe file to NO. This is recommended for asymmetrical configurations, where the exchange of computer functionalities is either not possible or not recommended.

EXIT STATUS

0	obschange terminated without error.
1	Invalid parameter or syntax error.
2	Command not allowed while OBSERVE not running.
4	Wrong system state: A function exchange is only possible, if one system has the OBSERVE state ORIGINAL and its partner system state BACKUP (or vice versa).
7	command aborted. Reason: Lost connection to partner system.
8	Function exchange aborted – system coming up with state OB lost connection to partner system.
9	Initialization error in message component.
20	Function exchange simultaneously initiated on both systems.
24	Command obschange not allowed (OBSCHANGE=NO).
255	Command not allowed in this operation mode.

FILES

/etc/default/observe
Environment variables for obschange.

SEE ALSO

observe(5) .

Fujitsu Siemens Computers Add-on Reference Manual

obsconfig(1M)

NAME

obsconfig - compile the OBSERVE configuration file

SYNOPSIS

obsconfig {-c [-r] | -s | -t [-r]} [-f *filename*]

DESCRIPTION

obsconfig compiles the OBSERVE configuration file /opt/observe/konfig/observe.konfig. After the compilation the new binary configuration file /opt/observe/konfig/konfig.bin is automatically copied to the partner system, provided that appropriate entries are made in .rhosts and the partner system is reachable. In order to start OBSERVE successfully both system must have identical copies of the binary configuration file. If rcopy fails a warning is output. In this case you must find a way to copy the binary configuration file manually to the partner system.

OPTIONS

-c A syntax and plausibility check is made of the configuration file. If no errors occur it will be compiled into the binary configuration file. Finally the binary is copied to the partner system.

-c -r
Same as option -c, but additionally the plausibility check is also done on the partner system. If the local plausibility check is successful and there is no connection to the partner system (via command rsh) then the remote plausibility check is also regarded as successful.

-s Execute only a syntax check.

-t Execute syntax and plausibility check.

-t -r
Same as option -t, additionally a plausibility check will be done on the partner system. If the local plausibility check is successful and there is no connection to the partner system (via command rsh) then the remote plausibility check is also regarded as successful.

-f *filename*
Use this option if the configuration file name differs from observe.konfig. The binary configuration file is named *filename*.bin.

ENVIRONMENT VARIABLES

The following environment variables enable you to choose the extent of the plausibility check:

PLAUSI_STANDARD	ON/OFF
PLAUSI_OBSERVERS	ON/OFF
PLAUSI_REACTION	ON/OFF
PLAUSI_HOSTS	ON/OFF
PLAUSI_LINES	ON/OFF
PLAUSI_TACLANS	ON/OFF
PLAUSI_SWITCHES	ON/OFF
PLAUSI_DEVICES	ON/OFF

These environment variables correspond to the components of the plausibility check. By setting these variables you can choose which plausibility checks shall be done and which not. The position of a switch ON/OFF determines whether the check result of the corresponding group counts or not. Is one of the switches set to OFF, then errors occurring within the corresponding group do not lead to an abortion of the command

processing. Is a specific switch set to `ON` and errors occur within the corresponding group then further processing is aborted. In absence of a switch the default value is `ON`. These environment variables are defined in `/etc/default/observe`.

EXIT STATUS

`0` `obsconfig` terminated without error.

`1` The syntax or plausibility check delivered an error – no binary configuration file could be created.

`2` Initialization error in message component.

FILES

`/opt/observe/konfig/config-sample.11`
Example OBSERVE configuration file.

`/etc/default/observe`
Environment variables for `obsconfig`.

SEE ALSO

`observe`(5) .

Fujitsu Siemens Computers Add-on Reference Manual

obsdown(1M)

NAME

obsdown - shut down computer or terminate OBSERVE

SYNOPSIS

obsdown [–h *hostname*] {–o | –s | –b | –u}

DESCRIPTION

With the obsdown command, you can

- terminate OBSERVE on both computers
- shut down the local computer
- shut down both computers
- terminate application programs and OBSERVE

OPTIONS

–h *hostname*
: Name of the host where the command should take effect. Default is the local host.

–o
: Terminate only the OBSERVE processes and observers on both OBSERVE cluster nodes. If the connection between the two nodes is down, obsdown –o can only be performed on the local system. obsdown –o does neither affect the operating system nor applications nor switches. The whole cluster continues running except for the OBSERVE processes. The security and monitoring provided by OBSERVE is, however, does no longer exist.

  obsdown –o is recommended for OBSERVE updates. This way you do not need to interrupt the cluster's operation for the update. To restart the OBSERVE processes after an update without switching peripherals and without restarting the applications see command startS98(1M) , –r option.

–s
: The local computer is shut down completely. OBSERVE invokes one of the scripts shutdown.orig, shutdown.backup or shutdown.both (according to the system's OBSERVE state) to terminate the applications. Then a system shutdown is initiated by calling the script shutdown.sys. This also triggers the failover of the local system's functions to its partner (if the failover is enabled on the partner system).

–b
: Both computers are shut down completely. This command can only be executed if OBSERVE is up and running on both computers. First OBSERVE invokes the appropriate script shutdown.orig or shutdown.backup on each system to terminate the applications, then a system shutdown is initiated by calling the script shutdown.sys. If the local computer currently has status OB (original and backup system), no shutdown is executed and the command is rejected with an error. This command should be used to shut down an OBSERVE cluster. The two systems are shut down together, thus both systems retain their OBSERVE state and functionality for the next startup.

  If you shut down the two computers separately (e.g. using obsdown –s), there is a danger that OBSERVE may interpret the shutdown of one computer as a system failure and initiate an undesired failover.

–u
: The applications and OBSERVE are terminated without being followed by a system shutdown. OBSERVE calls one of the scripts shutdown.orig, shutdown.backup or shutdown.both according to the OBSERVE state and then exits. This also triggers the failover of the local system's functions by the partner system (provided the failover is enabled on the partner system).

EXIT STATUS

0
: obsdown terminated without error.

1 Invalid parameter – syntax error.

2 Command not allowed while OBSERVE not running.

4 Shutdown not allowed in this system state, for example: `obsdown –b` is not allowed in state `OB`.

9 Initialization error in message component.

255 Command not allowed in this operation mode.

SEE ALSO

`startS98`(1M), `observe`(5).

Fujitsu Siemens Computers Add-on Reference Manual

obsforbid(1M)

NAME

`obsforbid -` forbid OBSERVE start

SYNOPSIS

`obsforbid`

DESCRIPTION

The `obsforbid` command prevents OBSERVE from being automatically started after the next reboot. This means, for example, that administrative or repair work can be carried out on the system without any interfering from OBSERVE (e.g. switchover of peripherals between computers).

EXIT STATUS

`0`	`obsforbid` terminated without error.
`1`	Invalid parameter or syntax error.
`7`	Command aborted – couldn't create file `/opt/observe/.internal/.obsstart`.
`9`	Initialization error in message component.
`255`	Command not allowed in this operation mode.

SEE ALSO

`obsallow`(1M), `observe`(5).

Fujitsu Siemens Computers Add-on Reference Manual

obshost(1M)

NAME

`obshost -` enable or (temporarily) disable system failover

SYNOPSIS

`obshost` [`–h` *hostname*] {`enable` | [`–t`] `disable`}

DESCRIPTION

This command `obshost` is used to enable or temporarily or permanently disable the failover of the partner system to the surviving local system in an OBSERVE cluster.

Disabled failover is effective only for normal operation of OBSERVE in 1:1 mode (one computer with original status, the other with backup status). If for example the original system fails and the failover is disabled on the backup system (command: `obshost disable`), the backup does not take over the functions of the failed original. Even if the failover is disabled OBSERVE still detects a failure of its partner system and reports the failure. If you re-enable the failover on one system (command `obshost enable`) while the other system is still down, the failover is initiated immediately (if OBSERVE already detected the failure).

`obsinfo(1M)` helps to find out whether the failover of the partner system is enabled or disabled.

If the parameter `autotake` has status `no` then the failover is disabled:

```
autotake:       status: no
```

If the parameter `autotake` has status `yes` then the failover is enabled:

```
autotake:       status: yes
```

OPTIONS

`–h` *hostname*
: Name of the host where the command should take effect. Default is the local host.

`enable`
: Enables the failover of a failed partner system (`autotake=yes`).

`disable`
: Failover of a failed partner system is disabled (remains disabled even after a reboot). If the partner system fails no failover will be initiated (`autotake=no`).

`–t disable`
: The option `–t` can only be combined with the keyword disable. The failover of a failed partner system is temporarily disabled until the next reboot (after the next reboot the failover is enabled again). If the partner system fails no failover will be initiated (`autotake=no`).

EXIT STATUS

`0` `obshost` terminated without error.

`1` Invalid parameter – syntax error.

`7` Command aborted. Reason: Lost connection to partner system.

`9` Initialization error in message component.

`255` Command not allowed in this operation mode.

SEE ALSO

`obsinfo(1M)` , `observe(5)` .

Fujitsu Siemens Computers Add-on Reference Manual

obsinfo(1M)

NAME

obsinfo - display system information

SYNOPSIS

obsinfo [–o]

obsinfo –l

DESCRIPTION

You can use the obsinfo command to display various system information:

- on the status of the two node OBSERVE cluster
- on the OBSERVE log file

You can process all information outputs with the standard Reliant UNIX tools (e.g. pg, vi) and also redirect them into files to be evaluated later.

OPTIONS

–o The current status of the OBSERVE cluster is displayed. The same output can be retrieved by invoking obsinfo without any option.

–l The OBSERVE log file is displayed.

EXAMPLES

```
obsinfo
               OBSERVE System Information (1:1)
============================================================
OBSERVE: started
------------------------------------------------------------
local system:  rm6a26                  status: backup
remote system: rm6a27                  status: orig
connection to remote host:             status: yes
control line 1 operating:              status: yes
control line 2 operating:              status: yes
start of observe allowed at reboot:    status: no
autorepair:                            status: no
autotake:                              status: yes
------------------------------------------------------------
```

EXIT STATUS

0 obsinfo terminated without error.

1 Invalid parameter – syntax error.

9 Initialization error in message component.

255 Command not allowed in this operation mode.

SEE ALSO

obsobjinfo(1M), observe(5).

Fujitsu Siemens Computers Add-on Reference Manual

obsobject(1M)

NAME

`obsobject -` control object monitoring

SYNOPSIS

`obsobject` [`–h` *hostname*] *option*

DESCRIPTION

The command `obsobject` is used to activate or deactivate the monitoring of individual objects or of all objects assigned to a specific observer.

Furthermore, you can also send user data to a specified observer. You can send user data either for all objects assigned to a specific observer by specifying the observer ID, or for one single object by specifying the object name – OBSERVE determines the corresponding observer automatically. The only user data accepted by the standard observers `EXIST` and `ALIVE` is the info string. This causes detailed information to be displayed on the observed objects. When writing your own user-defined observer, you can define any string of ASCII characters as "private string" to identify a specific function within your observer. These "private strings" are kind of commands that are executed either for the specified object or all objects of this observer when such a private string is sent to the observer.

Each object to be monitored by OBSERVE must be configured in the configuration file.

OPTIONS

`–h` *hostname*
: Specify the host where the command should take effect. Default is the local host.

`–ai` *observer_id*
: Activate monitoring of all objects assigned to the specified observer; *observer_id* is the observer ID entered in the configuration file.

`–di` *observer_id*
: Deactivate monitoring of all objects assigned to the observer defined by the specified observer ID.

`–an` *objectname*
: Activate monitoring of the object specified by the object name.

`–dn` *objectname*
: Deactivate monitoring of the object specified by the object name.

`–pi` *observer_id* [*string*]
: Call the callback function `info_fn` and, if applicable, send the user data specified in *string* (ASCII string; no more than 255 characters) to the observer designated by the observer.

`–pn` *objectname* [*string*]
: Call the callback function `info_fn` and, if applicable, send the user data specified in *string* to the observer which is monitoring the object designated by the object name (no more than 255 characters).

EXIT STATUS

`0` `obsobject` terminated without error.

`1` Invalid parameter – syntax error.

`2` Command not allowed while OBSERVE not running.

`7` Command aborted. Reason: Lost connection to partner system.

`9` Initialization error in message component.

`10` Unknown *observer_id*.

`11` Object monitoring not activated.

12 Object unknown or object not currently monitored.
13 Observer not started.
14 Objects are not monitored in this system state.
15 A command is currently active for this observer.
16 Observer not ready to receive.
17 Unknown object.
21 No objects configured for this observer.
22 A reaction script is active due to a state change.
255 Command not allowed in this operation mode.

SEE ALSO

obsinfo(1M), obsobjinfo(1M), observe(5).

Fujitsu Siemens Computers Add-on Reference Manual

obsobjinfo(1M)

NAME

obsobjinfo - display object information

SYNOPSIS

obsobjinfo [–h *hostname*] *option*

DESCRIPTION

The obsobjinfo command displays the status either of all the monitored objects assigned to a specified observer or of a selected, monitored object.

OPTIONS

–h *hostname*
: Name of the host where the command should take effect. Default is the local host.

–ii *observer_id*
: Output the status of all objects monitored by the observer specified by the given *observer_id*.

–in *objectname*
: Output the status of monitored object with the specified object name. The output information has the same structure as the information on all objects assigned to an observer.

EXAMPLES

```
# obsobjinfo -ii 1
                    Status of controlled object(s)
=========================================================================
observer name: obsexist
observer id:   1
=========================================================================
object name:  init
period:       5
-------------------------------------------------------------------------
                local                       remote
-------------------------------------------------------------------------
observation:    activated                   activated
status class:   1                           1
status number:  0                           0
status name:    RUNNING                     RUNNING
=========================================================================
object name:  silsd
period:       5
-------------------------------------------------------------------------
                local                       remote
-------------------------------------------------------------------------
observation:    activated                   activated
status class:   1                           1
status number:  0                           0
status name:    RUNNING                     RUNNING
-------------------------------------------------------------------------
```

EXIT STATUS

0 obsobjinfo terminated without error.

1 Invalid parameter – syntax error.

2 Command not allowed while OBSERVE not running.

7 Commando aborted. Reason: Lost connection to partner system.
9 Initialization error in message component.
10 Unknown *observer_id*.
11 Object monitoring not activated.
12 Unknown object or object not currently monitored.
13 Observer not started.
14 Objects not monitored in this system state.
15 A command is active for this observer.
16 Observer not ready to receive.
17 Unknown object.
21 No objects configured for this observer.
22 A reaction script is active due to a state change.
255 Command not allowed in this operation mode.

SEE ALSO

obsinfo(1M), observe(5).

Fujitsu Siemens Computers Add-on Reference Manual

obsress(1M)

NAME

`obsress -` start or stop resources

SYNOPSIS

`obsress` –`s`|–`k` [–`d` *usr_dir*] –`u` *usr_script ress0* [*ress1 ... ressN*]

DESCRIPTION

The command `obsress` allows to start or stop a single resource or a list of resources.

OPTIONS

–`s` Start resource.

–`k` Stop resource.

–`d` *usr_dir*
Directory where *usr_script* resides. Default is `/opt/observe/reactions/usr`.

–`u` *usr_script*
User reaction script for this kind of resource.

OBSERVE contains the following predefined standard user reaction scripts:

`010gmirror`
Activate/deactivate mirror disk groups.

`020filesystem`
Check and mount/unmount file systems.

`021fuserkill`
Terminate (kill) processes which use certain files or file systems.

`022share`
Share/unshare file systems with remote hosts.

`023nfs_filesystem`
Mount/unmount NFS remote file systems.

`025ip`
Configure/deconfigure network interfaces.

`025obsnetif`
Configure/deconfigure network interfaces via OBSNET.

`026obsnetdied`
Inform OBSNET clients of a server failure.

`030database`
Start/stop database.

`040xprint`
Start/stop Xprint.

`040spool`
Start/stop SPOOL.

`045networker`
Start/stop Networker domain.

`050application`
Start/stop applications.

ress0 ...
Resource to be started or stopped. The user defines these resources in the OBSERVE parameter file `/opt/observe/reactions/param.usr`.

To start or stop a single resource use the resource name as parameter to obsress, example `/home` to address the resource `/home` of the resource type `FILESYSTEMS`.

You can divide any resource type into different resource sets – one set for the original system, another for the backup and a third for the original and backup and if needed one for stand alone OBSERVE operation mode OBP.

If you want to start or stop such a set for the original system for instance, you can use the parameter `O_FILESYSTEMS` instead of a resource list. All resources belonging to the original resource set will then be started or stopped.

Example: Let the file systems used by the original system be `/home1`, `/home2`, `/home3`. The entry in `param.usr` is `O_FILESYSTEMS="/home1 /home2 /home3"`. The command `obsress –s -u 020filesystem O_FILESYSTEMS` starts all three file systems belonging to `O_FILESYSTEMS`. `obsress` checks and mounts these file systems in the same order as given in `param.usr`: `O_FILESYSTEMS="/home1 /home2 /home3"`. The command `obsress –k –u 020filesystem O_FILESYSTEMS` stops the resource set `O_FILESYSTEMS`. But in this case `obsress` uses the opposite order to unmount the file systems: `/home3`, `/home2`, `/home1`.

EXIT STATUS

`0` No error occurred when starting/stopping the resource.

R If an error occurs during the start/stop of a resource the exit code *R* of the user reaction script will be returned. In case of a list or set of resources processing will be aborted as soon as the first error occurs if the error handling parameter `IGNORE_`*resource_type*`_ERRORS` is set to `no`. If it is set to `yes` errors will be ignored and processing continues.

FILES

`/opt/observe/reactions/usr-samples/README.usr-samples`
This file explains the usage of the sample user reaction scripts in directory `/opt/observe/reactions/usr`.

`/opt/observe/reactions/usr-samples/README.*`
These README files explain how to configure the different resources.

SEE ALSO

`observe(5)` .

Fujitsu Siemens Computers Add-on Reference Manual

obsrestart(1M)

NAME

`obsrestart -` restart failed observer

SYNOPSIS

`obsrestart` [`–h` *hostname*] *observer_id*

DESCRIPTION

The command `obsrestart` restarts an observer that has failed.

OPTIONS

`–h` *hostname*
: Name of the host where the command should take effect. Default is the local host.

observer_id
: ID of the observer to be restarted. *observer_id* is a positive integer (as defined in the configuration file).

EXIT STATUS

`0`	`obsrestart` terminated without error, i.e. the restart has been initiated.
`1`	Invalid parameter – syntax error.
`9`	Initialization error in message component.
`10`	Unknown *observer_id*.
`11`	No objects assigned to this observer for monitoring.
`14`	Objects are not monitored in this system state.
`15`	Command currently active for this observer.
`16`	Observer not ready to receive.
`255`	Command not allowed in this operation mode.

SEE ALSO

`obsobject`(1M), `obsobjinfo`(1M), `observe`(5).

Fujitsu Siemens Computers Add-on Reference Manual

obsreturn(1M)

NAME

obsreturn - reestablish normal operation after failover

SYNOPSIS

obsreturn [–o | –b]

DESCRIPTION

The command obsreturn is issued on a system which has both OBSERVE states original and backup and on which the autorepair function has not been activated. Depending on the option either the original or backup functionality is. The partner system must be functioning and OBSERVE must be started and waiting to get back its functionality.

OPTIONS

–o The original system's functions are transferred.

–b The backup system's functions are transferred.

EXIT STATUS

0 obsreturn terminated without error.

1 Invalid parameter or syntax error.

2 Command not allowed – OBSERVE not running.

4 Wrong system state: Either OBSERVE is not in state OB (original and backup system), or OBSERVE is not RUNNING.

9 Initialization error in message component.

23 Wrong system state of partner system: OBSERVE on partner system is not in state STA_WAIT_REPAIR or STA_ASK.

24 obsreturn –o not allowed if partner has fixed state BACKUP (OBSCHANGE=NO).

obsreturn –b not allowed if partner has fixed state ORIGINAL (OBSCHANGE=NO).

In these cases obsreturn would force a function exchange, which is prohibited by OBSCHANGE=NO.

255 Command not allowed in this operation mode.

FILES

/etc/default/observe

In this file the environment variables AUTOREPAIR=yes|no and OBSCHANGE=yes|no can be set.

SEE ALSO

observe(5) .

Fujitsu Siemens Computers Add-on Reference Manual

obsstate(1M)

NAME

`obsstate -` define or change a computer's OBSERVE status

SYNOPSIS

`obsstate` [`–o` | `–b` | `–ob`]

DESCRIPTION

The obsstate command is used to define the OBSERVE status that will apply from the next OBSERVE start. Possible states are original system (`O`), backup system (`B`) or original and backup system (`OB`).

The command can also be used to change the computer's OBSERVE status, but it is rejected if OBSERVE is already running (see exit status 5).

OPTIONS

`–o` The computer is given status `O` (original system).

`–b` The computer is given status `B` (backup system).

`–ob` The computer is given status `OB` (original and backup system).

EXIT STATUS

`0` `obsstate` terminated without error.

`1` Invalid parameter or syntax error.

`5` Command not allowed while OBSERVE running.

`7` Command aborted – couldn't open file `/opt/observe/.internal/status`.

`9` Initialization error in message component.

`255` Command not allowed in this operation mode.

SEE ALSO

`observe(5)` .

Fujitsu Siemens Computers Add-on Reference Manual

oswiraid(1M)

NAME

`oswiraid -` switch RAID boxes or EMC² Symmetrix DRAID

SYNOPSIS

`oswiraid` `–z|–w` `–d` [`–l` *lun*] `–R` `–c` *chn* `–h` *otherhost raid-dev*

`oswiraid` `–s` `–c` *chn raid-dev*

DESCRIPTION

This command is used to switch RAID devices (of the following RAID box types: PXRC, PXRE, EMC² Symmetrix) or EMC² DRAID devices (EMC² Symmetrix only).

The first format serves for switching, the second format for status inquiry.

OPTIONS

`–z|–w`
: Specify option `–z` if you want to switch the RAID box to the local host and option `–w` if you want to switch the RAID box to the partner system.

`–d`
: Turn on diagnosis.

`–l` *lun*
: Number of the logical unit (LUN) to be switched. This parameter is not evaluated – all Logical Units of a RAID box are switched together.

`–c` *chn*
: This option specifies the switch position. Valid parameter values are 1 or 2. The switch position must be consistent with the option `–z|–w` and the OBSERVE configuration file. Channel 1 must be connected to the first host in the order of the HOST blocks of the OBSERVE configuration file. `oswiraid` does not perform any consistency checks.

`–R`
: Reinitialize the SCSI bus.

`–h` *otherhost*
: Name of the partner host.

`–s`
: Status inquiry.

raid-dev
: Node name of the specified RAID controller. This parameter must be given in the same format as displayed in the output of the command `autoconf –l`.

EXIT STATUS

`0`
: Command terminated successfully.

R
: If an error occurs `oswiraid` returns the exit code *R* of the invoked system command `/usr/bin/rdswitch` or `/sbin/draidd`.

SEE ALSO

`draidd`(1M), `oswiscu2`(1M), `oswisusi`(1M), `oswitaclan`(1M), `observe`(5), `autoconf`(8), `rdswitch`(8).

Fujitsu Siemens Computers Add-on Reference Manual

oswiscu2(1M)

NAME

`oswiscu2 -` actuate or query SISO or SCU switches

SYNOPSIS

Format 1

`oswiscu2` [`–v`] [`–O`] {`–z` | `–w`} `–c` *chn* `–p` *pbay* `–m` *module* {*port* | *drive@port* | *port1+port2*}

Format 2

`oswiscu2` [`–v`] `–p` {*pbay* | `a`} `–m` {*module* | `a`} `–s`

DESCRIPTION

This command switches SISO, SCU1, or SCU2 switches in peripheral cabinets and configures (deconfigures) connected (disconnected) devices.

Format 1 serves for switching, and **Format 2** for status inquiries to retrieve switch positions.

OPTIONS

Format 1

`–v` Verbose messages.

`–O` The optimize option is used by OBSERVE during the startup phase for switch optimization. If a switch is found in the desired switch position no additional switching is done.

You must not use this parameter when you invoke `oswiscu2` manually.

`–z` | `–w`

Use option `–z` to switch the SCSI devices (e.g. disks) residing in a switchable peripheral cabinet to the local system. After the switch command has terminated successfully, `oswiscu2` invokes `reinitdev(8)` to configure the devices into the operating system.

Use option `–w` to switch the devices to the partner system. After the switch command has terminated, `oswiscu2` invokes `changedev(8)` to deconfigure the corresponding devices.

`–c` *chn*

Defines the switch position (which channel). The channel parameter can be 1 or 2. The switch position must be consistent with the option `–z`|`–w` and the OBSERVE configuration file. The first host (according to the order of the HOST blocks in the OBSERVE configuration file) must be connected to channel 1. `oswiscu2` does not perform any consistency checks.

`–p` *pbay*

Number of the peripheral cabinet to be switched. The parameter value is an integer. The range depends on the version of the EIP (environmental interface processor) -> EIP2: 1..15 / EIP3: 0..63.

`–m` *module*

Number of the switchable module contained in the peripheral cabinet. The parameter value is an integer in the range from 1 to 9.

port Port for `reinitdev` or `changedev`.

drive@port

Drive for `changedev` and port for `reinitdev`.

port1+port2

In an asymmetric configuration with different port names *port1* refers to the first host from the HOST block in the OBSERVE configuration file and *port2* to the second.

Format 2

`–p` {*pbay* | `a`}

Either the number of the peripheral cabinet or `a` for all cabinets. The parameter value is an integer. The range depends on the version of the EIP (environmental interface processor) -> EIP2: 1..15 / EIP3: 0..63.

`-m` {*module* | `a`}
Either the number of the switchable module (in the range from 1 to 9) or `a` for all modules of the peripheral cabinet (`-p` *pbay*) or of all cabinets (`-p a`).

`-s` Display the result of the status inquiry, i.e. either the switch positions

- of all modules (option `-m a`)
- or of the specified module (option `-m` *module*) of the peripheral cabinet *pbay* (option `-p` *pbay*)
- or of all cabinets (option `-p a`).

EXIT STATUS

`0` Command terminated successfully.

R If an error occurs `oswiscu2` returns the exit code *R* of the underlying system commands `/etc/opt/havail/switchscu2` and `/etc/opt/havail/reinitdev` or `/etc/opt/havail/changedev` (invoked by `oswiscu2`).

SEE ALSO

`oswiraid`(1M), `oswisusi`(1M), `oswitaclan`(1M), `observe`(5), `changedev`(8), `reinitdev`(8), `switchscu2`(8).

Fujitsu Siemens Computers Add-on Reference Manual

oswisusi(1M)

NAME

`oswisusi -` actuate or query SUSI switches

SYNOPSIS

`oswisusi` [`–d`] `–a` *addr* `–c` *chn tty-dev*

`oswisusi –s` *tty-dev*

DESCRIPTION

The first format switches a SUSI switch, which is connected to control line *tty-dev*.

The second format determines the switch position of a SUSI switch.

OPTIONS

`–d` Turn on diagnosis.

`–a addr`
Addresses the SUSI module to be switched. The parameter value is an integer ranging from 1 to 10.

`–c chn`
This option determines the switch position. Valid parameter values are 1 or 2.

`–s` With option `–s oswisusi` will do a status inquiry and display the switch position.

tty-dev
Device node name of the serial interface which is connected to the control line of the corresponding SUSI switch module.

EXIT-CODES

`0` Command terminated successfully.

R If an error occurs `oswisusi` returns the exit code *R* of the underlying system command `/etc/opt/havail/switchsusi` (invoked by `oswisusi`).

SEE ALSO

`oswiraid`(1M), `oswiscu2`(1M), `oswitaclan`(1M), `observe`(5), `switchsusi`(8).

Fujitsu Siemens Computers Add-on Reference Manual

oswitaclan(1M)

NAME

`oswitaclan -` switch TACLAN devices

SYNOPSIS

`oswitaclan` *taclan-hostname port-list address* [*hostname*]

DESCRIPTION

This command serves for TACLAN switching. The TACLAN is logically switched by reparameterizing the TACLAN terminal server such that all connected terminals are either switched to the one system or to the other system.

OPTIONS

taclan-hostname
: Hostname of the TACLAN terminal server, whose ports are to be switched.

port-list
: List of port numbers, which are to be switched. Port numbers are integer values in the range of 1 to n (n depends on the number of ports the TACLAN model supports). Ports are separated by the character "@", for example "1@3@5@7".

address
: Internet address (in dot notation) of the host, to which the devices connected to the ports are assigned.

hostname
: Currently not evaluated.

EXIT STATUS

`0`
: Command terminated successfully.

R
: If an error occurs `oswitaclan` returns the exit code R of the underlying system command `/opt/adm863/na` (invoked by `oswitaclan`).

SEE ALSO

`oswiraid`(1M), `oswiscu2`(1M), `oswisusi`(1M), `observe`(5).

Fujitsu Siemens Computers Add-on Reference Manual

probe(1M)

NAME

`probe` - record a Probe

SYNOPSIS

`/opt/diag.d/bin/probe` –c *component* –o *output* [–i *interval* –r *repeat*] [–n *boardno*]

`/opt/diag.d/bin/probe` [–h]

`/opt/diag.d/bin/probe` [–?]

DESCRIPTION

The `probe` command is available to obtain a probe status information of a component or a board. The data recorded by `probe` contains both information from the software structures as well as the contents of the available hardware control and error register. This command enables you to write a snapshot of the current status of the addressed component to a diagnostic file *output*. As these probes are component-specific, `probe` must be provided separately for each component. The `–h` option can be used to ascertain for which components the `probe` command is implemented. The structure of the information provided by the `probe` command is component-dependent and is stored in the diagnostic file as a sequence of *Probe* parts.

OPTIONS

`–c` *component*

Refers to a system component. This can be a controller board in which case the board type (e.g. `sih`) must then be specified as the argument *component*. The board type is identical to the string which the command `autoconf(8)` also displays without the board number.

However, `probe` jobs can also be copied from other components (e.g. a port monitor such as `muxmon`). In this case, the argument *component* refers to the component name from which the information is to be obtained. The names of kernel components which support the `probe` command are permissible here. The `–h` option can be used to ascertain which components these are.

Specification of the component is mandatory.

`–o` *output*

Refers to the diagnostic file to which the data provided by `probe` are written. It is important to note that the command does not verify whether there is sufficient space in the file system where this file is located.

Specification of the component is mandatory.

`–i` *interval*

Determines the interval in seconds between 2 probes. This option only takes effect if a repeat value greater than 1 has been set with the `–r` option.

`–r` *repeat*

Specifies a repeat counter. The default value for this counter is 1. The repeat counter specifies how often a probe is to be taken. The interval between 2 probes is set by means of the `–i` option. If an `–i` option is not being used, the repeat counter is reset to 1 regardless of the value specified with the `–r` option.

`–n` *boardno*

Refers to the logical board number if the component is a controller board. The `probe` command does not check whether or not a board with this number exists. A *Probe* part, which may be empty, is always generated.

If this option is not used, a *Probe* part is generated for each logical board that is available on the system. The logical board numbers are recorded in the individual part headers.

The number of *Probe* parts is entered in the diagnostic file header `dh_pparts` and the logical board

numbers are marked in `dh_boardno` in bits.

If the *component* component is not a board, only a *Probe* part is generated and the *boardno* is ignored.

`-h` This argument is used to display on which boards or components the `probe` command can be used. The output is sent to `stdout`.

If this option is used, all of the other options are irrelevant.

`-?` This argument is used to display the user interface of the `probe` command on `stderr`. The return value is 0.

DIAGNOSTICS

If an error was not identified while the program was running, the program aborts with 0. Error messages are output on `stderr`. The following error codes are provided:

`E_ILLOPT`	1	illegal user interface on command
`E_ILLARG`	2	illegal arguments used
`E_PERM`	3	permission denied to uid
`E_CMDDIR`	4	No stat on components directory
`E_SYSTEM`	5	Can't call sh-command
`E_LIST`	6	List command with error
`E_OUTPUT`	7	Can't work with output file
`E_FILE`	8	Can't work with input file
`E_MAGIC`	9	wrong magic number

FILES

`/opt/diag.d/components`
`/usr/include/sys/diag.d/diaghead.h`

SEE ALSO

`autoconf(8)` .

Fujitsu Siemens Computers Add-on Reference Manual

rcvm(1M)

NAME

`rcvm -` start Reliant Cluster Visual Manager

SYNOPSIS

`rcvm` [*options*]

DESCRIPTION

The `rcvm` command starts the RCVM (Reliant Cluster Visual Manager), the graphical user interface that provides an interactive display of all Reliant Monitor resources, represented by RCVM icons called images, and their current states. The RCVM allows the user to temporarily manipulate a resource and to monitor the Reliant Monitor configuration using pull-down menus, commands, and graphic displays.

The `rcvm` command is usually run without any command line options; default values are appropriate for most configurations.

OPTIONS

`–display` *address*
: Specify the IP address of the terminal on which the GUI is to be displayed.

`–objectBackground` *color*
: Specify the background color of the images displayed in the Objects panel of the top-level window.

`–operationBackground` *color*
: Specify the background color of the operation or function buttons in the Operations panel or the Functions panel of the top-level window.

`–unknownBackground` *color*
: Specify the background color of the images when they are in the `Unknown` state.

`–onlineBackground` *color*
: Specify the background color of the images when they are in the `Online` state.

`–offlineBackground` *color*
: Specify the background color of the images when they are in the `Offline` state.

`–faultedBackground` *color*
: Specify the background color of the images when they are in the `Faulted` state.

`–waitBackground` *color*
: Specify the background color of the images when they are in the `Wait` state.

`–catchupBackground` *color*
: Specify the background color of the images when they are in the `CatchUp` state.

`–deactBackground` *color*
: Specify the background color of the images when they are in the `Deact` state.

`–pixmapPath` *path*
: Specify the path where the pixmap files used by `rcvm` are installed. (Only when the path is being changed from the default path, `/usr/lib/X11/pixmaps/Rvm`.)

`–bitmapPath` *path*
: Specify the path where the bitmap files used by `rcvm` are installed. (Only when the path is being changed from the default path, `/usr/lib/X11/bitmaps/Rvm`.)

`–logFile` *name*
: Specify the path of the log file to be used by `rcvm`. (Only when changing the default RCVM log file name, `/var/opt/reliant/log/guiDlog.`*userid*. This option is used for debugging.)

`–rsrcTypesDb` *name*

Specify the path of the resource types database file to be used by `rcvm`. (Only when using a customized resource types database file.)

`–rsrcStatesDb` *name*
Specify the path of the resource states database file to be used by `rcvm`. (Only when using a customized resource states database file.)

`–dbgMsg` [*value*]
Specify whether the debugging messages should be displayed; *value* can be either `True` or `False`. The default is `True` when this option is specified on the command line. (This option is used for debugging.)

`–logMsg` [*value*]
Specify whether the I/O messages should be logged to the log file; *value* can be either `True` or `False`. The default is `True` when this option is specified on the command line. (This option is used for debugging.)

`–visualBell` [*value*]
Specify whether the visual bell should be turned on when any notice/warning/error messages are received by `rcvm` from the configuration monitor, `cm`; *value* can be either `True` or `False`. The default is `True` when this option is specified on the command line.

`–resourceImage` [*value*]
Specify whether to display images in resource images or state images; *value* can be either `True` or `False`. The default is `True` when this option is specified on the command line.

`–help`
Usage is displayed.

EXAMPLES

The following command starts the RCVM using the default settings:

```
rcvm
```

Fujitsu Siemens Computers Add-on Reference Manual

sihadm(1M)
RM400 only

NAME

`sihadm -` manage SIH monitor

SYNOPSIS

`/usr/sbin/sihadm –d` *admdev* `–a` *adjust* [`–l` *ttylab* `–k` *linklab*] [`–p` '*cmd*']

`/usr/sbin/sihadm –I` [*svctag*] *pmtag*

`/usr/sbin/sihadm –R` [*svctag*] *pmtag*

`/usr/sbin/sihadm –S` [*svctag*] *pmtag*

`/usr/sbin/sihadm –T` *level pmtag*

`/usr/sbin/sihadm –v` *type*

`/usr/sbin/sihadm –V`

`/usr/sbin/sihadm –h|–?`

DESCRIPTION

The `sihadm` command aims to facilitate the administration of SIH monitors [see `sihmon(1M)`]. The Service Access Facility (SAF) requires that such a command be made available for each monitor. The administration table for the SIH monitor (`_pmtab`) is administered by the Service Access Controller commands (SAC commands) `sacadm(1M)` and `pmadm(1M)` . In addition to its other functions, `sihadm` generates formatted output for these commands.

In the following, an SIH driver link is understood as a port in terms of the SAF. The link configuration is defined by an entry in the `_pmtab` file. An entry of this kind is identified by a unique name (SIH monitor service ID).

The `sihadm` command contains the following functions:

- Formatted, SIH monitor-specific output
- Setting loading mode in SIH driver (`–I`)
- Reseting loading mode in SIH driver (`–R`)
- Status output of links monitored by the SIH monitor (`–S`)
- Setting of SIH monitor trace level (`–T`)
- Output of expected version IDs of the correctly integrated programs and configuration files (`–V` and `–v`)
- Usage output (`–h` and `–?`)

If used correctly, all output is sent to `stdout`. Error messages or the use of commands are output to `stderr`.

The use of the `–I`, `–R`, `–S` and `–T` options is only possible with `root` authorization.

OPTIONS

`–d` *admdev*

This option is used to define the absolute or relative path name link-specific admin devices of the SIH driver.

`–a` *adjust*

This option defines the time and the form of parameter settings for the SIH monitor. For the *adjust* argument, exactly one of the following characters is allowed:

`c` Each time a HDLC device is switched on, and following a change in the SIH monitor status to `enabled`, the current HDLC parameters are read from the SIH driver and stored.

Then the default HDLC parameters are set and the `'`*cmd*' loading program is invoked. Following successful completion of the loading process, the previously stored HDLC parameters are

returned to the SIH driver (`c`urrent).

The default HDLC parameters are, thus, only used during the loading process. The current HDLC parameters are retained. This setting is only useful for loadable HDLC devices.

`e` Each time a HDLC device is switched on, and following the change in status of the SIH monitor to `enabled`, the default HDLC parameters are set (`e`ver).

`o` The default HDLC parameters are set when the status of the SIH monitor changes to `enabled`. Under normal operating conditions, this occurs only once when the SIH monitor is started on (`o`nce).

`n` The default HDLC parameters are never set (`n`ever). In this case, the `-l` and `-k` options are superfluous.

`-l` *ttylab*

This option provides a flag which refers to an entry in `/etc/ttydefs`. This entry informs the SIH of the line speed. Specification of the option is mandatory when the `-ac`, `-ae` or `-ao` options are used.

`-k` *linklab*

This option provides a link flag which refers to an entry for link-specific HDLC parameters in `/etc/hdlcdefs`. Specification of the option is mandatory when the `-ac`, `-ae` or `-ao` options are used.

`-p` '*cmd*'

This option provides the complete call for a loading program for connected HDLC devices. As the loading program options and arguments must be distinguished from the `sihadm` options, the *cmd* argument must be contained in inverted commas (').

Specification of the option is mandatory when the `-ac` option is used.

`-I` [*svctag*] *pmtag*

This option is used to set the loading mode of an SIH driver's admin device. The required admin device is taken from the `_pmtab` table of the SIH monitor *pmtag* with the SIH monitor service ID *svctag*.

If *svctag* is omitted, all entries in the `_pmtab` table are taken into account.

Loading mode is *not* set if the relevant link is already being monitored by an SIH monitor. In this case, a warning is output to `stderr`.

`-R` [*svctag*] *pmtag*

This option is used to reset the loading mode of an SIH driver's admin device. The required admin device is taken from the `_pmtab` table of the SIH monitor *pmtag* with the SIH monitor service ID *svctag*.

If *svctag* is omitted, all entries in the `_pmtab` table are taken into account.

Loading mode is *not* reset, if the relevant link is already being monitored by an SIH monitor. In this case, a warning is output to `stderr`.

`-S` [*svctag*] *pmtag*

This option is used to output to `stdout`, the status of the *svctag* link being monitored by the SIH monitor *pmtag*. If *svctag* is omitted, the status of this and all other links known to the SIH monitor is output.

The command requests the relevant SIH monitor to write the `/var/saf/`*pmtag*`/state` status file. The required information is taken from this file.

`-T` *level pmtag*

This option can be used to change the trace level of the SIH monitor called *pmtag*. The argument *level* indicates the trace level:

`0` no outputs (trace switched off, default)

`1` output of error messages and important operating messages

`2` additional output of detailed operating messages

`-V` This option outputs the version ID for the format of the `_pmtab` file, which is expected for the corresponding version of the `sihadm` and `sihmon(1M)` commands (see options `-va` and `-vm`).

Output is sent to `stdout` without line feed. It is identical to the output of the `sihadm –vp` command.

`–v` *type*

This option provides the version ID of the correctly integrated programs and configuration files. The *type* argument can assume one of the following values:

`a` Output of `sihadm` version ID

`m` Output of expected `sihmon` version ID

`p` Output of expected `_pmtab` version ID

`t` Output of expected `ttydefs` version ID

`d` Output of expected `hdlcdefs` version ID

`l` Output of expected `hdlclabs` version ID

`x` Block output of the above-listed version IDs

The `sihadm` and `sihmon` version IDs are identical to the version ID of the corresponding `SIsih` package.

With the exception of the `-vx` option, all output is sent to `stdout` without line feed.

`–h` | `–?`

Returns information about the use of the `sihadm` command to `stderr`.

pmtag

Name of an SIH monitor incarnation.

svctag

Name of an SIH monitor `_pmtab` entry, which describes a link that is being monitored by the SIH monitor (SIH monitor service ID).

NOTES

The formatted SIH monitor-specific output and the output of the `–V` command are executed *without* line feed. They are particularly suitable for creating an SIH monitor incarnation using `sacadm –a`, and for generating `_pmtab` entries using `pmadm –a`. If the `sihadm` command `–p` option is *not* used when `pmadm –a` is invoked, the activated SIH monitor provides the corresponding HDLC parameter setting, but does not initiate loading.

The `–S` and `–T` options communicate with the current SIH monitor incarnation *pmtag*. *The corresponding SIH monitor process must, therefore, be already started*.

The `–I` and `–R` options influence later communication between an SIH monitor incarnation and the SIH driver. *A corresponding SIH monitor process must not be started*.

When the system is started, the `–I` option should be called for all links to be monitored *before* the SAC is started. This will ensure that the SIH driver is in loading mode before a user channel (multi-device) is opened at the SIH driver.

The default HDLC parameters consist of line and link-specific HDLC parameters from the `hdlcdefs` file and are recorded with the help of the link tag, admin device and `hdlclabs` file.

DIAGNOSTICS

The `sihadm –S` *svctag pmtag* call output has the following format:

`service tag` *svctag* `--` *internal_state* [*reason*]

The *internal_state* field indicates the internal status of the *svctag* link. The following entries are possible:

`ENABLED`

The link is being monitored. No errors have occurred (normal).

`LOADING`

Loading is currently in progress for the link.

`ONRETRY`

A process has failed for the link (e.g. the loading of an HDLC device with a workstation program) and will be repeated when a specific interval has expired [see `sihmon(1M)`]. An explanation can be found

in the following *reason* error text.

DISABLED
: Monitoring of the link has been terminated. An explanation can be found in the following *reason* error text. This type of link can be reactivated by invoking `sacadm -x -p`*pmtag* when the cause of the error has been eliminated.

When the `sihadm -S` *pmtag* call is used, the status of the SIH monitor incarnation is output with the current option settings. All SIH monitor service IDs are numbered.

If no error occurs, `sihadm` terminates with the return value 0. In the event of an error, the return values have the following meanings:

`1` = Incorrect option argument used (`A_WRARG`)

`2` = Incorrect combination of options (`A_ILLOPT`)

`3` = Unknown SIH monitor name *pmtag* (`A_UPMTAG`)

`4` = SIH monitor *pmtag* is not started (`A_NOTRUN`)

`5` = Command not executed as `root` (`A_NOROOT`)

`6` = Unknown *admdev*, or no access rights (`A_NODEV`)

`7` = SIH monitor has written no `state` file (`A_NOSTATE`)

`8` = *svctag* not contained in `_pmtab` or `disabled` (`A_NOSERV`)

`9` = `_pmtab` cannot be opened (`A_NOPMTAB`)

`10` = Incorrect `_pmtab` version (`A_WPMTAB`)

`11` = SIH monitor *pmtag* is active (`A_ISRUN`)

`12` = Incorrect SIH monitor program name (`A_WRPNAME`)

`50` = Internal error (`A_INTERNAL`)

FILES

`/usr/lib/saf/sihmon`

`/etc/saf/`*pmtag*`/_pmtab`

`/etc/ttydefs`

`/etc/hdlcdefs`

`/var/sadm/terms/hdlclabs`

`/var/saf/`*pmtag*`/state`

SEE ALSO

`pmadm`(1M), `sac`(1M), `sacadm`(1M), `sihmon`(1M).

Fujitsu Siemens Computers Add-on Reference Manual

sihmon(1M)
RM400 only

NAME

sihmon - SIH monitor

SYNOPSIS

/usr/lib/saf/sihmon [–d *level*] [–f *file*] [–t *timeout*] [–r *retry*] [–s *size*]

/usr/lib/saf/sihmon –v

/usr/lib/saf/sihmon –h |–?

DESCRIPTION

The SIH monitor sihmon administers HDLC devices, which are connected to an SIH board via an SIH driver. It sets the HDLC parameters and ensures that a deactivated, loadable HDLC device is loaded with a workstation program when switched on.

The SIH monitor is a port monitor in terms of the Service Access Facility (SAF) and uses the latter's facilities to configure and control the monitor. It is designed to run under the control of the Service Access Controller (SAC). The sacadm(1M) command informs the SAC of a SIH monitor incarnation. The pmadm(1M) and sihadm(1M) commands are used to make entries and changes in the SIH monitor (_pmtab) administration table. Additional SIH monitor-specific commands are available using the sihadm(1M) command.

An SIH driver link should be understood as a port in the SAF sense. The configuration of a link is defined by an entry in the _pmtab file. This kind of entry is identified by a specific name (SIH monitor service ID).

Messages are usually written to the /var/saf/*pmtag*/log log file of the relevant SIH monitor incarnation. Usage output and error messages resulting from incorrect application of the options are output to stderr. Other error messages, which are issued before the log file can be opened, are output to both stderr and the console terminal.

The SIH monitor can only be successfully started with root authorization.

OPTIONS

If sihmon is invoked without options, the following default values apply.

–d *level*

This option presets the SIH monitor trace level. The *level* argument shows the trace level:

0 no output (trace not activated, default value)

1 output of error messages and important operating messages

2 additional output of detailed operating messages

The trace level can be changed during operation using the sihadm(1M) command. If the –f option is not used, all trace outputs are added to the end of the /var/saf/*pmtag*/trace file.

–f *file*

With this option, the *file* file is used as the trace file instead of the default /var/saf/*pmtag*/trace file. An absolute path name is expected. Traces are only output if the current trace level is 0 (see –d option).

–t *timeout*

This option specifies the period of time in seconds which should expire before a failed operation (e.g. loading error) is repeated. If a value lower than 10 is specified, 10 is used. The default value is 90 seconds.

–r *retry*

This option specifies how many times in succession an operation can fail (e.g. loading error), before the corresponding link is *internally* DISABLED. An internally DISABLED link can be reactivated by invoking sacadm –x –p*pmtag* again. The internal status of the link can be established by invoking sihadm –S.

The *retry* argument can assume 0 or a positive value. A 0 value allows unlimited repetition (default value).

−s *size*
: This option defines the maximum size of the log file /var/saf/*pmtag*/log in bytes. If this size is exceeded, the file is copied to the /var/saf/*pmtag*/log.alt file. The original file is shortened to zero length.

  The value for the *size* argument may be 0 or positive as long as it does not exceed the maximum file size authorized by the system. If the 0 value is selected, no data is written to the log file. Values between 1 and 1023 are increased to 1024. The default value is 1048576 bytes (1 MB).

−v
: Outputs the SIH monitor version ID to stdout without line feed. This ID is identical to the version ID of the SIsih package.

−h | −?
: The command syntax is displayed on stderr.

NOTES

The SIH monitor can only make link-specific contact with the SIH driver, if the SIH driver for this link is switched to loading mode *before* the corresponding multidevice is opened for the first time. This can be done by correctly timing the start of the SIH monitor incarnation, or by correctly timing the invocation of the sihadm −I command (e.g. when the system is being brought up).

It is assumed that all links entered in the _pmtab file of the corresponding SIH monitor incarnation are to be monitored. Thus, for entries in the _pmtab file which are disabled, the corresponding SIH driver admin device is opened and loading mode is set. Other operations are started, when the entry is enabled.

The loading mode for an admin device is reset when an entry is removed from the _pmtab file, or the SIH monitor terminates.

An SIH monitor incarnation can monitor a maximum of 200 links. Thus, the _pmtab file should only contain a maximum of 200 enabled entries at any one time.

For internal purposes the SIH monitor has a maximum requirement of 15 file descriptors and one additional file descriptor for each link to be monitored.

All log and trace files are protected from overflow. If the file size exceeds the maximum value, the file is copied to a file with the same prefix name and the suffix .alt. The original file is reduced to zero length. The prescribed maximum values are:

/var/saf/*pmtag*/log
: see option -s

/var/saf/*pmtag*/trace
: 2097152 bytes (2 MB)

/var/saf/*pmtag*/state
: 2097152 bytes (2 MB)

Loading programs invoked from the SIH monitor [see sihadm(1M) and xbload(1M)], should observe the following conventions:

1. The loading program should show a return value of 0 if the relevant HDLC device has been successfully loaded or was already loaded. Otherwise, it should display a positive return value; the loading attempt is repeated when the timed interval has expired (see the −t and −r options).
2. Error messages should be written to a separate file or to stderr.

If changes are made to the /var/sadm/terms/hdlclabs, /etc/hdlcdefs or /etc/ttydefs files, which affect the SIH monitor incarnation *pmtag*, the SIH monitor must be notified of the changes by the sacadm −x −p*pmtag* command. The changes are then taken into account the *next* time HDLC parameters are set. (see the −a option of the sihadm(1M) command).

DIAGNOSTICS

For diagnostic purposes, the SIH monitor administers the private `/var/saf/`*pmtag*`/log` log file (see the `-s` option), to which exceptional events and error messages are written. A line in the log file has the following format:

type dd`.`*mm HH*`:`*MM*`:`*SS pid text*

The *type* field can contain the following entries:

`INFO` The line contains important information (e.g. start and stop of the SIH monitor). There is no error.

`WARN` An error has occurred, which can result in one or more links not being monitored. Such messages should not appear during normal operation.

`FATAL`
A fatal error has occurred which leads to the immediate termination of the SIH monitor.

dd, *mm*, *HH*, *MM* and *SS* give the day, month, hour, minute and second of the error message.

The *pid* field shows the process number of the corresponding SIH monitor incarnation.

The *text* field describes the error that has occurred.

The format of the trace file entries is identical to the format of the log file entries. If the current trace level is positive, all log messages are also written to the trace file. This is also the case when the SIH monitor incarnated was started using the `-s0` option.

All log messages are also written to Logging 3.0 (Component number 1080005). The error level can accept the values 6 (`INFO`), 4 (`WARN`) or 3 (`FATAL`). Each log messages has a different number. The format of the logging text is shown below:

pid text

Under normal operating conditions, the SIH monitor is terminated using the SAC `SIGTERM`(15) signal. It is, however, also possible to terminate the monitor manually using the `kill -15`*pid*. The return value in both cases is 0. The `SIGKILL`(9) signal should be avoided as a means to terminate the monitor, as clean-up operations cannot be executed.

In the case of an error, the SIH monitor terminates with a `FATAL` error message in the log file. In this case, the return value is one of the following:

`1` = Wrong option used (`E_ILLOPT`)

`2` = Wrong argument for an option (`E_WRARG`)

`3` = No `root` authorization (`E_NOROOT`)

`4` = Monitor incarnation already running (`E_PIDLOCK`)

`5` = `_pid` cannot be created (`E_CPID`)

`6` = *pmtag* variable not available in the process environment (`E_PMTAG`)

`7` = Log file cannot be opened (`E_NOLOG`)

`8` = Content of `ISTATE` variables unknown (`E_ISTATE`)

`9` = `_pmpipe` cannot be opened (`E_PMPIPE`)

`10` = `_sacpipe` cannot be opened (`E_SACPIPE`)

`11` = Too few file descriptors available in the system (`E_TOOFEWFDS`)

`12` = `_pmtab` cannot be opened (`E_NOPMTAB`)

`13` = `hdlclabs` cannot be opened (`E_NOHDLCLABS`)

`14` = `hdlcdefs` cannot be opened (`E_NOHDLCDEFS`)

`15` = `ttydefs` cannot be opened (`E_NOTTYDEFS`)

`16` = Wrong `_pmtab` version (`E_WRPMTAB`)

`17` = Wrong `hdlclabs` version (`E_WRHDLCLABS`)

`18` = Wrong `hdlcdefs` wrong (`E_WRHDLCDEFS`)

`19` = Wrong `ttydefs` version (`E_WRTTYDEFS`)

20 = Error in polling _pmpipe file (E_POLLSAC)
21 = Error in loader exec (E_CLD_EXEC)
50 = Internal program error (E_INTERNAL1)
99 = Internal program error – core dump (E_INTERNAL2)

FILES

/usr/sbin/sihadm
/etc/saf/*pmtag*/_pmtab
/etc/ttydefs
/etc/hdlcdefs
/var/sadm/terms/hdlclabs
/var/saf/*pmtag*/log[.alt]
/var/saf/*pmtag*/trace[.alt]
/var/saf/*pmtag*/state[.alt]

SEE ALSO

pmadm(1M), sac(1M), sacadm(1M), sihadm(1M), xbload(1M).

Fujitsu Siemens Computers Add-on Reference Manual

soft_get(1M)
Cluster Operating System

NAME

soft_get - request a special file from all known nodes

SYNOPSIS

soft_get [–C *clos_name*] [–r *node*] [–p *prefix*] –n *file* [–n *file*]

soft_get –?

DESCRIPTION

The soft_get command is used to request the specified files from all nodes in the selected CLOS subsystems [see clos(7)]. The files are transferred synchronously and in parallel with the assistance of the soft_put(1M) command on every node. No guarantee is given, however, that the transmission will be completed within a given time.

The nodes are checked to make sure they are accessible. If a node cannot be reached at present, a recovery script is produced for this node and distributed to all other accessible nodes.

The –n *file* arguments are the base names [see basename(1)] of the files that are to be distributed. The absolute pathname of the files is made up of the absolute pathname of the *prefix* argument for the –p option, followed by a directory with the local node name [see uname(1)] and the appended base name of the –n option. The default *prefix* is the /etc/default/soft-DB directory. Each node distributes a different file based on this method for deriving the pathname. The CLOS sync_cmd(1M) command is used in this case.

OPTIONS

–C *clos_name*
: The name of a cluster, in which files are to be requested, can be selected in the *clos_name* argument with this option. If this option is not specified, files will be requested from all nodes in all clusters.

Warning: The $CLOSNAME environment variable is not evaluated.

–r *node*
: Recovery mode.

In this mode, files are only transmitted for the *node* node in the case of a recovery. The node name for the absolute pathname of the file(s) is passed in the RECOVERPARAM environment variable. The command does not transfer any file and ends immediately without errors if this environment variable is not set.

–n *file*
: Base name of a file.

The –n option is used to specify the base name of a *file* file from which the complete pathname on the selected nodes is derived.

–p *prefix*
: Transfer a prefix for the absolute pathname.

If files that are not located under the default /etc/default/soft-DB directory are to be distributed, this option can be used to choose another path. The paths of all files (–n option) begin with this *prefix*.

–? Usage of this command.

DIAGNOSTICS

If the soft_get command discovers an error situation locally before a file is requested, this is reported and the file is not requested on any of the nodes. The return value is 0 if no error is discovered. If an error is reported, the return value is as follows:

1 "`filename missing`", "`illegal option`": Incorrect usage. At least one file name must be specified.

3 "`sync_cmd failed`". The actual `sync_cmd`(1M) command for requesting a file with the `soft_put`(1M) argument has failed. It has been verified that all nodes are accessible for this command and that inaccessibility is not the cause of the problem. Insufficient resources would cause the command to be repeated and is hence also not a reason for the command to fail.

4 "`you need to be superuser!`". This command can only be executed by a superuser.

5 "`NODE not available for recovery!`". The node specified in recovery mode (`-r` option) is not accessible.

FILES

`/etc/default/soft-DB`

`/etc/default/soft-DB`*nodename*

SEE ALSO

`basename`(1), `uname`(1), `soft_put`(1M), `sync_cmd`(1M), `clos`(7).

Fujitsu Siemens Computers Add-on Reference Manual

soft_put(1M)
Cluster Operating System

NAME

soft_put - distribute a special file to all known nodes

SYNOPSIS

soft_put [–C *clos_name*] [–r *node*] [–p *prefix*] –n *file* [–n *file*]

soft_put –?

DESCRIPTION

The soft_put command is used to distribute the specified files to all nodes in the selected CLOS subsystems [see clos(7)]. The files are transferred synchronously and in parallel. No guarantee is given, however, that the transmission will be completed within a given time.

soft_put distributes the file(s) via the node channels [see clos(7)] and waits for acknowledgement from each specified node. The CLOS command sync_send(1M) is used in this case.

The nodes are checked to make sure they are accessible. If s node cannot be reached at present, a recovery script is produced for this node and distributed to all other accessible nodes.

The –n *file* arguments are the base names [see basename(1)] of the files that are to be distributed. The absolute pathname of the files is made up of the absolute pathname of the *prefix* argument for the –p option, followed by a directory with the local node name [see uname(1)] and the appended base name of the –n option. The default *prefix* is the /etc/default/soft-DB directory. Each node distributes a different file based on this method for deriving the pathname.

OPTIONS

–C *clos_name*
: The name of a cluster, in which files are to be distributed, can be selected in the *clos_name* argument with this option. If this option is not specified, all nodes in all clusters will be selected for distributing files.

Warning: The $CLOSNAME environment variable is not evaluated.

–r *node*
: Recovery mode.

In this mode, files are only transmitted for the *node* node in the case of a recovery. The node name for the absolute pathname of the file(s) is passed in the RECOVERPARAM environment variable. The command does not transfer any file and ends immediately without errors if this environment variable is not set.

–n *file*
: Base name of a file.

The –n option is used to specify the base name of a *file* file from which the complete pathname is derived.

–p *prefix*
: Transfer a prefix for the absolute pathname.

If files that are not located under the default /etc/default/soft-DB directory are to be distributed, this option can be used to choose another path. The paths of all files (–n option) begin with this *prefix*.

–? Usage of this command.

DIAGNOSTICS

If the soft_put command discovers an error situation locally before a file is distributed, this is reported and the file is not copied to any of the nodes. The return value is 0 if no error is discovered. If an error is reported,

the return value is as follows:

1 "`filename missing`", "`illegal option`": Incorrect usage. At least one file name must be specified.

2 "`file` *filename* `not available`". A file with the specified base name does not exist in the specified directory. No error is reported in recovery mode (`-r` option).

3 "`sync_send failed`". The actual `sync_send`(1M) command for transmitting a file has failed. It has been verified that all nodes are accessible for this command and that inaccessibility is not the cause of the problem. Insufficient resources would cause the command to be repeated and is hence also not a reason for the command to fail.

4 "`you need to be superuser!`". This command can only be executed by a superuser.

5 "`NODE not available for recovery!`". The node specified in recovery mode (`-r` option) is not accessible.

FILES

`/etc/default/soft-DB`

`/etc/default/soft-DB`*nodename*

SEE ALSO

`basename`(1), `uname`(1), `sync_send`(1M), `clos`(7).

Fujitsu Siemens Computers Add-on Reference Manual

spsadmin(1M)

NAME

`spsadmin` - SPS cluster administration command

SYNOPSIS

`spsadmin` [*option* ...]

DESCRIPTION

The only function `spsadmin` currently has is to disable or enable SPS cluster functionality which takes effect the next time the system boots. If the operation was successful, it prints a message to stdout; otherwise, it prints an error message to stderr.

OPTIONS

`–d` Disable SPS functionality. Changes take effect at next boot.

`–e` Enable SPS functionality. Changes take effect at next boot.

`–q` Query SPS status for next boot.

EXIT STATUS

`0` The operation was completed successfully.

`!=0` The operation was not completed successfully.

Fujitsu Siemens Computers Add-on Reference Manual

SPSverify(1M)

NAME

`SPSverify -` verify SPS installation and configuration

SYNOPSIS

`SPSverify [–v` *level* `][–V][–h]`

DESCRIPTION

Verification of SPS installation and configuration can be initiated at any time with the `SPSverify` command. The output produced depends on the "state" of the system; but even at the "lowest" state it should indicate whether the cluster node is adequately configured to be part of the cluster. This may give rise to different scenarios:

– only the software packages are installed:
`SPSverify` determines the consistency of the installation and system dependencies. If no error occurs, it should report that the system is ready for configuration.

– the cluster node is configured but SPS is not running:
`SPSverify` checks the correctness of the local configuration. If the result is positive, it contacts the other cluster nodes (if it's not the first node in the cluster) via "raw" interconnect access to retrieve the total cluster configuration. It then checks the correctness of the configuration for the entire cluster. If no error occurs, it should report that the system is ready to be part of the cluster. Furthermore, it lists all nodes found thus far within the cluster.

– the cluster node is already integrated within the cluster:
`SPSverify` performs all the checks above and reports the results (on an active cluster, no failures are expected). Furthermore, it checks for potential problems using the cluster interconnect.

Because `SPSverify` contacts `CCfgD`(1M) on remote nodes in a cluster to retrieve overall configuration information, the installation and bring-up phase should be performed on an individual node basis, for example configure, verify and start the first node in a cluster, then the second node in a cluster, etc.

`SPSverify` provides a short descriptive report of detected failures. Details are logged in a temporary `/tmp/sps.log.$pid` file. The name of this file, which contains additional diagnostics, is also displayed.

OPTIONS

`–v` *level*
Enable verbose mode at the specified numeric level. The default level is `1` (moderate diagnostic output). Currently only level `0` has an effect, e.g. only failures are reported.

`–V` Print version and exit.

`–h` Print usage and exit.

FILES

`/tmp/sps.log.$pid`

SEE ALSO

`CCfgD`(1M), `CCfgCmd`(1M).

Fujitsu Siemens Computers Add-on Reference Manual

startS98(1M)

NAME

`startS98 -` start OBSERVE

SYNOPSIS

`startS98 [ –m | –r ]`

DESCRIPTION

With the command `startS98` you can either start OBSERVE manually or initiate a restart of the OBSERVE processes only (without switching peripherals or starting applications). The OBSERVE startup script `/etc/rc2.d/S98observe` will be called with the appropriate parameters.

OPTIONS

`–m` Start OBSERVE manually.

If OBSERVE was started either manually via the command `startS98` or via the OBSERVE administrator's menu and the current state is not `OB` and it is unable to contact its partner system, then OBSERVE will ask the administrator:

```
-------------------------------------------------------------
OBSERVE:  (1) Start as original and backup system
OBSERVE:  (2) Terminate Observe
Enter your choice [1|2]:
OBSERVE:  If the partner system is booting please wait.
-------------------------------------------------------------
```

If you want OBSERVE to come up as `OB` (original and backup), then you enter (1). This overrides a previously set different state and OBSERVE will come up as `OB`. This is recommended if the partner system is still down and will stay down for a while until it gets repaired.

If you do not want OBSERVE to come up right now, you can abort the startup by choosing (2).

If you wish to take full advantage of the OBSERVE observation and failover facilities you can start OBSERVE on the partner system and wait until the partner system becomes ready. This is recommended when the partner system is booting [after a repair] and coming up itself.

`–r` Restart OBSERVE processes without applications.

The OBSERVE restart mechanism helps to continue operation during an OBSERVE update installation.

First you terminate the OBSERVE processes and observers on both systems by using `obsdown –o` (the applications keep running), then you install the new OBSERVE version and finally you restart the OBSERVE processes and observers by using `startS98 –r`. After a restart you enjoy again the full OBSERVE functionality and you didn't have to stop the applications during the update.

During an OBSERVE restart you may encounter the following situations:

```
-----------------------------------------------------------------------
OBSERVE: The system is waiting for its partner now.
         Please start OBSERVE by "startS98 -r" on the partner system too
         If you want to terminate OBSERVE, please enter (q):
-----------------------------------------------------------------------
```

To achieve full OBSERVE protection restart OBSERVE on the partner system either via the command `startS98 –r` or via the OBSERVE administrator's menu. As soon as the partner system is restarted and a connection between the two systems is established OBSERVE comes up on both systems in normal operation mode.

Choose (q) if you want to abort the OBSERVE restart.

If you use the restart option although you previously shut down OBSERVE and all applications, you get the following warning:

```
-------------------------------------------------------------------------
WARNING: During the last OBSERVE shutdown the applications
         have been stopped, too.
         A full start of OBSERVE is recommended now -> (1).
         OBSERVE restart would neither switch the peripherals
         nor start the applications again -> (2).
         (1) OBSERVE full start
         (2) OBSERVE restart only (without applications)
         (q) Quit
             Enter your choice [1|2|q]:
-------------------------------------------------------------------------
```

If you do not use the restart option although you previously shut down OBSERVE without terminating the applications (`obsdown -o`), you get the following warning:

```
-------------------------------------------------------------------------
WARNING: OBSERVE has been stopped last time without terminating
         the applications.
         Therefore only the OBSERVE processes should be restarted -> (1).
         A full start of OBSERVE would switch the peripherals and
         restart the applications again.
         If the applications have not been stopped meanwhile manually
         a full start will lead to severe inconsistencies -> (2).
         (1) OBSERVE restart only (without applications)
         (2) OBSERVE full start
         (q) Quit
             Enter your choice [1|2|q]:
-------------------------------------------------------------------------
```

EXIT STATUS

0 `startS98` could successfully perform the OBSERVE startup.

1 An error occurred (see file `/opt/observe/log/usr_log`).

SEE ALSO

`obsdown`(1M), `observe`(5).

Fujitsu Siemens Computers Add-on Reference Manual

statistic(1M)

NAME

`statistic -` record statistics

SYNOPSIS

`/opt/diag.d/bin/statistic` `–c` *component* `–o` *output* `–i` *interval* [`–n` *boardno*] [`–r` *repeat*]

`/opt/diag.d/bin/statistic` [`–h`]

`/opt/diag.d/bin/statistic` [`–?`]

DESCRIPTION

The `statistic` command is provided to ascertain statistics of a component or a board. As these statistics are component-specific, `statistic` must be made available for each individual component. The `–h` option can be used to ascertain the components for which the `statistic` command can be implemented. The structure of the information provided by the `statistic` command is also component-dependent and is stored in the diagnostic file as a sequence of *Statistic* parts.

OPTIONS

`–c` *component*

Refers to a system component. This can be a controller board in which case the board type (e.g. `sih`) must be specified as the argument *component*. The board type is identical to the string which the `autoconf(8)` command also displays.

However, `statistic` jobs can also be copied from other components (e.g. a port monitor such as `muxmon`). In this case, the argument *component* refers to the component name from which the information is to be obtained. The names of kernel components that support the `statistic` command are permitted here. The `–h` option can be used to determine which components these are.

Specification of the component is mandatory.

`–o` *output*

Refers to the diagnostic file to which the data provided by `statistic` is written. It is important to note that the command does not check whether there is sufficient space in the file system where this file is located.

Specification of the file is mandatory.

`–i` *interval*

determines the period in seconds in which the statistics are collected.

Specification of this time period is mandatory.

`–r` *repeat*

Specifies a repeat counter. The default counter is set to 1. The repeat counter specifies how often a statistic is to be taken. The time interval between 2 statistic measurements is 0.

`–n` *boardno*

Refers to the logical board number if the component is a controller board. The `statistic` command does not check whether a board with this number exists. A *Statistic* part, which may be empty, is always generated.

If this option is not used, a *Statistic* part is generated for each logical board that is available on the system. The logical board numbers are recorded in the respective part headers.

The number of the *Statistic* parts is entered in the diagnostic file header `dh_sparts` and the logical board numbers are marked in `dh_boardno` in bits.

If the component *component* is not a board, only a *Statistic* part is generated and the *boardno* is ignored.

`–h` This option is only used to display on which boards or components the `statistic` command can be

used. The output is sent to `stdout`.

If this option is used, all other options are irrelevant.

`-?` This option is used to display the user interface of the `statistic` command on `stderr`. The return value is 0.

DIAGNOSTIC

If an error has been identified while the program is running, the program aborts with 0. Error messages are output on `stderr`. The following error codes are provided:

`E_ILLOPT`	1	illegal user interface on command
`E_ILLARG`	2	illegal arguments used
`E_PERM`	3	permission denied to uid
`E_CMDDIR`	4	No stat on components directory
`E_SYSTEM`	5	Can't call sh-command
`E_LIST`	6	List command with error
`E_OUTPUT`	7	Can't work with output file
`E_FILE`	8	Can't work with input file
`E_MAGIC`	9	wrong magic number

FILES

`/opt/diag.d/components`
`/usr/include/sys/diag.d/diaghead.h`

SEE ALSO

`autoconf`(8) .

Fujitsu Siemens Computers Add-on Reference Manual

sync_cmd(1M)
Cluster Operating System

NAME

sync_cmd - synchronously executing a command in the cluster

SYNOPSIS

sync_cmd [−C *clos_name*] [−a] [−T] [−t *taglist*] [−z *sec*] *cmd_string*

sync_cmd −?

DESCRIPTION

Using the sync_cmd command, a CLOS subsystem [see clos(7)] can execute a command on all specified and accessible nodes in the cluster. The command is executed synchronously and in parallel on all nodes. There is no guarantee that the command will be executed within a specific period of time. sync_cmd distributes the command over the node channels [see clos(7)] and waits for acknowledgements.

The selected nodes are checked to determine their accessibility. If a node is selected that is not currently accessible, the command terminates with an error if the −a option is not set. File transfer only commences when all nodes can be accessed or A mode is enabled [see the −a option]. If a command could not be executed on one node or if transmission of a file to a node fails, an appropriate flag is set in the return value of the command.

A one-line acknowledgement is sent to standard output for every selected node. If the command was executed successfully, the acknowledgement reads:

NODE *tag*: Command done! exit status *number*

The *cmd_string* argument represents a shell command string that can be executed under Reliant UNIX. This string is distributed via the Closmon monitor [see closmon(1M)] to the selected nodes where it is executed. To execute the command, the callers' default shell is started on every node in their $HOME directory with the appropriate access rights. The $PATH, $HOME and $USER environment variables are taken from the /etc/passwd file of the respective node. The command return value is sent to standard output with the *number* status.

OPTIONS

−C *clos_name*
: The name of the cluster is selected in the *clos_name* argument using this option. If this option is not specified, the name from the $CLOSNAME environment variable is used or, if this is not set, the name is taken from the /var/clos/.CLOSdefault file.

−a
: Enabling A mode (do not abort).

  If this mode is enabled, the command is not aborted if one of the selected nodes cannot be accessed. In this case, an appropriate message is output for the node and the accompanying flag is set in the return value.

−t *taglist*
: Selecting the target node(s).

  The *taglist* argument is a list of tag names for the target nodes on which the command is executed. A list consists of tag names, separated by commas. A white space character (blank, tab, newline character) concludes the list. If this option is not used, all configured nodes are on the list.

  The command is executed on all nodes on the *taglist* list. If one of the nodes on the list was not accessible when the command was called and if A mode was not enabled, sync_cmd terminates with an error.

  If the −T option was used, the command is executed on all nodes *except* those on the list.

−T
: Inverting the target node list.

The list of target nodes specified with the `-t` option is inverted. This means that the *cmd_string* command is executed on all nodes that are *not* on the list. If the list is blank (no `-t` option), the command is executed on all nodes except the one on which `sync_cmd` was called.

`-z` *sec*

Enabling time monitoring (timeout).

If you wish, you can specify a timeout which is given in *sec*. This means that the `sync_cmd` command is ended and the commands already started are aborted up to *sec* seconds after the last command was started. The return value has the value 8.

`-?` Usage of this command.

DIAGNOSTICS

If the `sync_cmd` command comes across a local error before the command is distributed, this is reported and the command is not executed on any of the nodes on the *taglist* list. The return value is 0 if no error was discovered. If an error is reported, the return value is to be interpreted as follows: Return values are less than or equal to 6. In these cases, no command is transmitted or started. A message is sent to standard error output and the return values have the following meaning:

`1` "`illegal option`", "`Command missing`". Incorrect call.

`2` "`local node not configured`". The local node is not configured in the cluster.

`3` "`local node not integrated`". The local node is not integrated in the cluster (`NOT_READY` or `NOT_AVAIL` status).

`4` "`Open channel to NODE` *tag* `fails!`" The specified node channel cannot be opened for the caller.

`5` "`Poll system call failed!`". The `poll(2)` system call fails.

`6` "`not enough space for poll array`". The program has insufficient address space.

If transmission and execution of the command has begun, confirmation is expected from each node and reported to standard output. If an error occurred on one of the node channels when transferring the file or before the command is executed, the acknowledgement text has the following format:

`NODE` *tag*`:` *acknowledgement text*`! errno` *number* `->` *error text*

The return value then contains the following values:

`8` "`*** NODE` *TAG*`: command interrupted`". A timeout or a signal aborted the commands. A corresponding message is sent to standard error output.

`16` "`Can't send cmd request`". The command could not be issued on the node channel.

`32` "`Wrong ID on CLOS_ACKN response`". `CLOS` did not process the commands correctly.

`64` "`POLLERR on channel to NODE` *tag*". A `poll(2)` error occurred on one of the channels.

`128` "`Command Transmission failed`". Transmission failed on one of the channels.

FILES

`/var/clos/.CLOSdefault`

`/etc/passwd`

SEE ALSO

`ksh(1)`, `sh(1)`, `closmon(1M)`, `poll(2)`, `passwd(4)`, `environ(5)`, `clos(7)`.

Fujitsu Siemens Computers Add-on Reference Manual

sync_send(1M)
Cluster Operating System

NAME

sync_send - synchronously transmitting a file in the cluster

SYNOPSIS

sync_send [−C *clos_name*] [−a] [−T] [−t *taglist*] [−i *icmd*] [−p *pcmd*] [−z *sec*] *file*

sync_send −?

DESCRIPTION

Using the sync_send command, a CLOS subsystem [see clos(7)] can transfer a file to all specified and accessible nodes in the cluster. The file is transferred synchronously and in parallel to all nodes. There is no guarantee that the file will be transferred within a specific period of time. sync_send distributes the file over the node channels [see clos(7)] and waits for acknowledgements from each specified node.

The selected nodes are checked to determine their accessibility. If a node is selected that is not currently accessible, the command terminates with an error (exception: A mode [see −a option]). File transfer only commences when all nodes can be accessed. If the transfer of a file to an individual node subsequently fails, an appropriate flag is set in the return value of the command.

A one-line acknowledgement is sent to standard output for every selected node. If the command was executed successfully, the acknowledgement reads:

NODE *tag*: Transfer one! exit status *number*

The *file* argument represents a pathname for the file to be transferred from the local node to the same position on the remote node. The file access rights are determined by the local Closmon monitor [see closmon(1M)] and are also transferred to the nodes. The file transferred belongs to the user who called the command and it is assigned the transferred access rights.

To execute the *icmd* and *pcmd* commands, the callers' default shell is started on every node in their $HOME directory with the appropriate access rights. The $PATH, $HOME and $USER environment variables are taken from the /etc/passwd file of the respective node.

OPTIONS

−C *clos_name*

The name of the cluster is selected in the *clos_name* argument using this option. If this option is not specified, the name from the $CLOSNAME environment variable is used or, if this is not set, the name is taken from the /var/clos/.CLOSdefault file.

−a Enabling A mode (do not abort).

If this mode is enabled, file transfer is not aborted if one of the selected nodes cannot be accessed. In this case, an appropriate message is output for the node and the accompanying flag is set in the return value.

−i *icmd*

Executing an *icmd* initial command.

The −i option is used to specify the pathname for an *icmd* command that is executed before the file is transferred. This command can be a script that ensures that data remains consistent when it is being transferred. Thus, a daemon could be forced to close the *file*.

A shell, which has access authorization set by the user calling the command, is created on the target nodes. This shell is responsible for executing the command.

−p *pcmd*

Executing a *pcmd* terminating command.

The −p option is used to specify the pathname for a *pcmd* command that is executed after the file is

transferred. For example, the daemon could be forced to reopen the *file*.

A shell which has access authorization set by the user calling the command is created on the target nodes. This shell is responsible for executing the command.

`-t` *taglist*

Selecting the target node(s).

The *taglist* argument is a list of tag names for the target nodes on which the command is executed. A list consists of tag names, separated by commas. A white space character (blank, tab, newline character) concludes the list. If this option is not used, all configured nodes are on the list. Invalid tag names are ignored.

The *file* is copied to all nodes on the *taglist* list. If one of the nodes on the list cannot be accessed when the command is called and if A mode is not enabled, `sync_send` terminates with an error.

If the `-T` option was used, the command is executed on all nodes *except* on those contained on the list.

`-T` Inverting the target node list.

The list of target nodes specified with the `-t` option is inverted. This means that the file is transferred to all nodes that are *not* on the list. If the list is blank (no `-t` option), the *file* is transferred to all nodes except the one on which `sync_send` was called.

`-z` *sec*

Enabling a timeout.

An optional timeout can be specified in *sec*. This means that the `sync_send` command will be terminated a maximum of *sec* seconds after the transfer has started. The return value contains the value 8.

`-?` Usage of this command.

DIAGNOSTICS

If the `sync_send` command comes across a local error before the file is distributed, this is reported and the file is not copied to any of the nodes on the *taglist* list. The return value is 0 if no error was discovered. If an error is reported, the return value is to be interpreted as follows: return values are less than or equal to 7. In these cases, no file is transmitted nor is a command started. A message is sent to standard error output and the meaning of the return values is as follows:

`1` "`illegal option`", "`file missing`". Incorrect call.

`2` "`local node not configured`". The local node is not configured in the cluster.

`3` "`local node not integrated`". The local node is not integrated in the cluster (`NOT_READY` or `NOT_AVAIL` status).

`4` "`Open channel to NODE` *tag* `fails!`" The specified node channel cannot be opened for the caller.

`5` "`Poll system call failed!`". The `poll`(2) system call failed.

`6` "`not enough space for poll array`". The program has insufficient address space.

`7` "`file dos not exist`". One of the specified file names does not exist.

If file transfer and execution of the command has begun, confirmation is expected from each node and is reported to standard output. If an error occurred on one of the node channels when transferring the file or before the command is executed, the acknowledgement text has the following format:

`NODE` *tag*`:` *acknowledgement text*`! errno` *number* `->` *error text*

The return value then contains the following values:

`8` "`Can't send requested file`", "`node not reachable`" or "`Can't open node channel`". The node channel could not be used since the node itself was not accessible. If the command was not called in A mode, it it aborted immediately.

"`***` `NODE` *TAG*`: command interrupted`". The timeout or a signal has aborted the command. An appropriate message is sent to standard error output.

16 "Wrong ID on CLOS_ACKN response" or "Wrong ID on CLOS_RETURN response". A log error occurred when transferring the file to a node. An invalid job was acknowledged.

32 "POLLERR on channel to NODE *tag*". A poll(2) error occurred on one of the channels.

64 "File Transmission failed". The node signals that an error occurred when transferring the file.

FILES

/var/clos/.CLOSdefault
/etc/passwd

SEE ALSO

ksh(1), sh(1), closmon(1M), poll(2), environ(5), clos(7).

Fujitsu Siemens Computers Add-on Reference Manual

taginfo(1M)
Cluster Operating System

NAME

`taginfo` - outputting information on the current system

SYNOPSIS

`taginfo` [`–asnrvmMphNt`]

`taginfo –?`

DESCRIPTION

`taginfo` writes information about the current operating system to standard output.

The format of the output is the same as that of the `uname(1)` command. The `taginfo` output differs from the `uname` output in that it has three additional fields and three corresponding options. The output format is as follows:

system node release version type capacity processor ident provider tagname

OPTIONS

No option is specified :
This is the same as specifying the `–s` option.

`–a` (a – all) Information is output about the operating system name, the system node name, the operating system version number, the operating system revision version, the machine type, the number of central units, the number of megabytes in main memory, the processor name, the hardware identification, the hardware manufacturer and the tag name in the cluster. Example:
`SINIX-Y utmost 5.43 B0032 RM600 6/256 R4000 SYS0064005 SNI node_1`

`–s` (s – system) The operating system name is output. The operating system is known under this name on the local installation. Example: `SINIX-Y`. This is the default setting if no option is specified.

`–n` (n – node) The operating system node name is output. The system can be addressed under this name within a network. Example: `utmost`.

`–r` (r – release) The operating system version number is output. Example: `5.43`.

`–v` (v – version) The operating system revision version is output. Example: `B0032`.

`–m` (m – machine type) The name of the machine type is output. Example: `RM600`.

`–M` (M – machine capacity) The number of processors is output as well as the number of megabytes in the main memory. Example: `6/256`.

`–p` (p – processor type) The processor family of the system with which the user is currently working is output. Example: `R4000` for RISC processors.

`–h` (h – hardware ident) If this option is selected, the hardware identification number is output. This number should be quoted when dealing with Customer Service. Example: `SYS0064005`.

`–N` (N – name of hardware provider) This option is used to output the name of the hardware manufacturer which, in most cases, is `SNI`.

`–t` (t – tag name) If the system belongs to the cluster specified using the `$CLOSNAME` environment variable or the `/var/clos/.CLOSdefault` file, the tag name is output. Example: `node_1`.

`–?` Usage of this command.

FILES

`/var/clos/.CLOSdefault`

SEE ALSO

uname(1), sysinfo(2), uname(2).

Xscon(1M)
SIDLM

NAME

`Xscon -` mapping consoles in an SIDLM cluster

SYNOPSIS

`/usr/bin/X11/Xscon` [`–a`] [`–l`] [*node* ...]

DESCRIPTION

`Xscon` enables consoles of the cluster nodes to be mapped to X windows on the Cluster Console. All configured nodes are offered in a multiple selection list. By clicking on the "Establish connections" button, an xterm window starts for every selected node's console.

`Xscon` is a front end for the `xscon` program. It sets some environment variables which customize `xscon` for the needs of `commd` (e.g. creating logfiles for every node's console in `/opt/SIdlm/commd`).

In addition, a `commdLog` window is created automatically, if it does not already exist. The `commdLog` window shows the logging output created by `commd`, which is also written into the file `/opt/SIdlm/commd/commdLog`.

In your applications desktop, there is an icon for `Xscon` and one for `commdLog` which means `Xscon -l`.

OPTIONS

`–a` Create console windows for all configured nodes instead of supplying a selection list.

`–l` Forces creation of a new `commdLog` window.

node A cluster node that is configured for `xscon` use in `/etc/uucp/Devices` and `/etc/uucp/Systems`.

FILES

`/opt/SIdlm/commd/commdLog`
`/etc/uucp/Devices`
`/etc/uucp/Systems`

SEE ALSO

`xscon`(1M) .

C Functions(3)

cd_defs(3)

NAME

`cd_defs -` set and get default values for user/group ID andfile/directory permissions

SYNOPSIS

```
#include <sys/cdrom.h>
int cd_defs(path, cmd, defs) char *path;
int cmd;
struct cd_defs *defs;
```

DESCRIPTION

This function sets or gets (based upon *cmd*) the defaults for user IDs, group IDs, file/directory permissions and directory search permissions for the mounted CD-ROM. The argument *path* points to a mount-point of a

CD-ROM file system. The argument *cmd* is either {CD_SETDEFS} or {CD_GETDEFS}. The declaration for struct cd_defs and the definitions for {CD_SETDEFS} and {CD_GETDEFSS} are contained in <sys/cdrom.h>.

RETURN VALUE

Upon successful completion, cd_defs() returns a value of zero.

In case of an error, –1 is returned and errno is set to indicate the error.

ERRORS

The cd_defs() function will fail if:

EACCES
: Search permission is denied for a component of the *path* prefix, or read permission is denied on the mount-point.

ENAMETOOLONG
: The length of the *path* string exceeds {PATH_MAX}, or a pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect.

ENOENT
: A component of *path* does not exist, or the *path* argument points to an empty string.

EINVAL
: The value of *cmd* or values of members of the cd_defs structure are invalid.

  The argument *path* does not point to a mount-point of a CD-ROM file system.

ENOTDIR
: A component of the *path* prefix is not a directory.

EFAULT
: The address of structure cd_defs or *path* is invalid.

EPERM
: User does not have appropriate privileges in case of setting values.

EINTR
: A signal was caught during the cd_defs() function.

EMFILE
: {OPEN_MAX} file descriptors are currently open in the calling process.

ENFILE
: The system file table is full.

APPLICATION USAGE

The setting of default values is restricted to a user with appropriate privileges. In case of setting default values this function is intended to be used only directly after the CD-ROM has been mounted, before any access to the CD-ROM is done. If the function is applied for setting default values when files or directories have already been opened the effect of this function on these files and directories is undefined.

Fujitsu Siemens Computers Add-on Reference Manual

cd_drec(3)

NAME

cd_drec, cd_cdrec - read Directory Record from CD-ROMdirectory

SYNOPSIS

```
#include <sys/cdrom.h>
int cd_drec(path, fsec, drec) char *path;
int fsec;
struct iso9660_drec *drec;
int cd_cdrec(path, fsec, drec) char *path;
int fsec;
char *drec;
```

DESCRIPTION

The `cd_drec()` function fills the `drec` structure with the contents of the Directory Record associated with a file or directory referred to by *path*. The argument *fsec* specifies the File Section of that file. The numbering starts with one. The number −1 denotes the last File Section of the named file, or the only File Section of the named directory. The argument *path* points to a file or directory within the CD-ROM file hierarchy. The declaration for `struct iso9660_drec` is contained in `<sys/cdrom.h>`.

The `cd_cdrec()` function copies the complete Directory Record as recorded on the CD-ROM to the address *drec*. The user must allocate {`CD_MAXDRECL`} bytes for the Directory Record. {`CD_MAXDRECL`} is contained in `<sys/cdrom.h>`.

RETURN VALUE

Upon successful completion, the functions return a value of zero. In case of an error, −1 is returned and `errno` is set to indicate the error.

ERRORS

The functions will fail if:

`EACCES`
: Search permission is denied for a component of the *path* prefix, or read permission is denied for the directory in which the file or directory pointed to by *path* is located.

`ENAMETOOLONG`
: The length of the *path* string exceeds {`PATH_MAX`}, or a pathname component is longer than {`NAME_MAX`} while {`_POSIX_NO_TRUNC`} is in effect.

`ENOENT`
: A component of *path* does not exist, or the *path* argument points to an empty string.

`ENOTDIR`
: A component of the *path* prefix is not a directory.

`EFAULT`
: The address of *drec* or *path* is invalid.

`EINVAL`
: The value of *fsec* is invalid.

  The argument *path* points to a file/directory that is not within the CD-ROM file hierarchy.

`ENXIO`
: The CD-ROM is not in the drive, or a read error occurred.

`EINTR`
: A signal was caught during one of the functions.

EMFILE
{OPEN_MAX} file descriptors are currently open in the calling process.

ENFILE
The system file table is full.

Fujitsu Siemens Computers Add-on Reference Manual

cd_getdevmap(3)

NAME

cd_getdevmap - get mappings of major/minor numbers

SYNOPSIS

```
#include <sys/cdrom.h>
int cd_getdevmap(path, pathlen, index, new_major, new_minor) char *path;
int pathlen;
int index;
int *new_major;
int *new_minor;
```

DESCRIPTION

This function gets the major and minor numbers of one device file on a mounted CD-ROM. The argument *path* points to a file or directory within the CD-ROM file hierarchy. The argument *index* refers to the *index*'th mapped device file. Mappings can be obtained by *path* or *index*.

If *index* is zero, this function gets the mapped major and minor numbers of the device file pointed to by *path*. The value of the mapped major number shall be returned in *new_major*. The value of the mapped minor number shall be returned in *new_minor*. The value of *pathlen* is not used.

If *index* is not zero, this function gets the major and minor numbers and pathname of the *index*'th mapped device file. Numbering for *index* starts at one. The value of the mapped major number shall be returned in *new_major*. The value of the mapped minor number shall be returned in *new_minor*. The pathname of the device file shall be returned in *path*. If the length of the pathname for the device file is longer than *pathlen* the pathname returned in *path* will be truncated to *pathlen* length and will not be NULL terminated.

RETURN VALUE

cd_getdevmap() will return the length of pathname if the device file is successfully returned (a return value of zero means mapping not found). Note: if the pathname is truncated, the return value will be larger than *pathlen*.

In case of error, –1 is returned and errno is set to indicate the error.

ERRORS

EACCESS
: Search permission is denied for a component of the *path* prefix or read permission on the device file pointed to by *path* is denied.

ENAMETOOLONG
: The length of the *path* string exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect.

ENOENT
: A component of *path* does not exist or the *path* argument points to an empty string.

ENOTDIR
: A component of the *path* prefix is not a directory.

EFAULT
: The address of *path*, *new_major*, or *new_minor* is invalid.

EINVAL
: The value of *index* or *pathlen* is invalid.

: The argument *path* points to a file/directory not within a CD-ROM file hierarchy.

: The file pointed to by *path* is not a device file.

`ENXIO`
The CD-ROM is not in the drive or a read error occurred.

`EINTR`
A signal was caught during the `cd_getdevmap()` function.

`EMFILE`
`{OPEN_MAX}` file descriptors are currently open in the calling process.

`ENFILE`
The system file table is full.

APPLICATION USAGE

The maximum number of mappings is defined in `<sys/cdrom.h>`. The device file mappings for a mounted CD-ROM are undone when the CD-ROM is unmounted.

The *index* numbers from 1 to *n* (where *n* is the number of the last device file mapping) are always guaranteed to have a device file mapping associated with the number. Thus if an application wishes to successively delete all device file mappings, one at a time, it would call `cd_getdevmap()` with *index* equal to 1, and then `cd_setdevmap()` with `CD_UNSETDMAP` in a loop until `cd_getdevmap()` returns zero.

Fujitsu Siemens Computers Add-on Reference Manual

cd_idmap(3)

NAME

`cd_idmap -` set and get mappings of user/group IDs

SYNOPSIS

```
#include <sys/cdrom.h>
int cd_idmap(path, cmd, idmap, nmaps) char *path;
int cmd;
struct cd_idmap *idmap;
int *nmaps;
```

DESCRIPTION

This function sets or gets (based upon *cmd*) the mapping of user or group IDs for the mounted CD-ROM. The argument *path* points to a mount-point of a CD-ROM file system.

If *cmd* is {`CD_SETUMAP`} or {`CD_SETGMAP`}, this uses the *idmap* array of mappings to map user or group IDs. The argument *nmaps* indicates the number of mappings in the array. Any mapping or value set with a previous invocation of `cd_idmap()` is overridden. When nmaps is zero, none of previously set mappings will stay in effect.

If *cmd* is {`CD_GETUMAP`} or {`CD_GETGMAP`}, this fills the array of mappings of User or Group IDs with the current mappings. On call, *nmaps* must contain the maximum number of mappings that may be returned. On return, *nmaps* will contain the number of mappings that are returned.

The declaration for `struct cd_idmap` and the definitions for {`CD_SETUMAP`}/{`CD_SETGMAP`} and {`CD_GETUMAP`}/{`CD_GETGMAP`} are contained in `<sys/cdrom.h>`.

RETURN VALUE

Upon successful completion, `cd_idmap()` returns a value of zero. In case of an error, −1 is returned and `errno` is set to indicate the error.

ERRORS

The `cd_idmap()` function will fail if:

`EACCES`
: Search permission is denied for a component of the *path* prefix, or read permission is denied on the mount-point.

`ENAMETOOLONG`
: The length of the *path* string exceeds {`PATH_MAX`}, or a pathname component is longer than {`NAME_MAX`} while {`_POSIX_NO_TRUNC`} is in effect.

`ENOENT`
: A component of *path* does not exist, or the *path* argument points to an empty string.

`EINVAL`
: The value of *cmd* or *nmaps* (negative or larger than {`CD_MAXUMAP`} and {`CD_MAXGMAP`}, respectively) is invalid.

  A member of the `cd_idmap` structure is invalid: *from_id* is larger than 65535 or a value in *to_uid* or *to_gid* is not supported by the system. Note that this error will not be given when *from_id* does not exist on the CD-ROM, or when *to_uid* is not defined in the User Database or *to_gid* is not defined in the Group Database.

  The argument *path* does not point to a mount-point of a CD-ROM file system.

`ENOTDIR`
: A component of the *path* prefix is not a directory.

EFAULT
: The address of structure `cd_idmap` or *path* is invalid.

EPERM
: User does not have appropriate privileges in case of setting values.

EINTR
: A signal was caught during the `cd_idmap()` function.

EMFILE
: {`OPEN_MAX`} file descriptors are currently open in the calling process.

ENFILE
: The system file table is full.

APPLICATION USAGE

The setting of values is restricted to a user with appropriate privileges. Only files and directories with an unrestricted final XAR are subject to this mapping. An owner or group identification of zero is not permitted by ISO 9660 to appear in an unrestricted XAR, and thus the behaviour is undefined if a mapping is given for the value zero. Use `cd_defs()` to change the default values. The maximum number of mappings is defined in `<sys/cdrom.h>`. In case of set mappings this function is intended to be used only directly after the CD-ROM has been mounted, before any access to the CD-ROM is done. If the function is applied for setting mappings when files or directories have already been opened the effect of the function on these files and directories is undefined.

Fujitsu Siemens Computers Add-on Reference Manual

cd_nmconv(3)

NAME

`cd_nmconv` - set and get CD-ROM file name conversion

SYNOPSIS

```
#include <sys/cdrom.h>
int cd_nmconv(path, cmd, flag) char *path;
int cmd;
int *flag;
```

DESCRIPTION

This function sets or gets (based upon *cmd*) the name conversion flag for file names on the mounted CD-ROM. The argument *path* points to a mount-point of a CD-ROM file system. The argument *cmd* is either {`CD_SETNMCONV`} or {`CD_GETNMCONV`}. The parameter *flag* is one of the following:

{`CD_NOCONV`}
: No conversion (default after mounting of the CD-ROM).

{`CD_LOWER`}
: Characters in Identifiers on CD-ROM are converted to lower case when represented in the XSI file hierarchy. If a File Identifier contains no File Name Extension, the SEPARATOR 1 (.) is not represented.

{`CD_NOVERSION`}
: The Version Number and the SEPARATOR 2 (;) of a File Identifier is not represented in the XSI file hierarchy.

{`CD_LOWER`} and {`CD_NOVERSION`} may be bitwise-inclusive or-ed.

RETURN VALUE

Upon successful completion, `cd_nmconv()` returns a value of zero. In case of an error, −1 is returned and `errno` is set to indicate the error.

ERRORS

The `cd_nmconv()` function will fail if:

`EACCES`
: Search permission is denied for a component of the *path* prefix, or read permission is denied on the mount-point.

`ENAMETOOLONG`
: The length of the *path* string exceeds {`PATH_MAX`}, or a pathname component is longer than {`NAME_MAX`} while {`_POSIX_NO_TRUNC`} is in effect.

`ENOENT`
: A component of *path* does not exist, or the *path* argument points to an empty string.

`EINVAL`
: The value of *cmd* or *flag* is invalid.

  The argument *path* does not point to a mount-point of a CD-ROM file system.

`ENOTDIR`
: A component of the *path* prefix is not a directory.

`EFAULT`
: The address of *flag* or *path* is invalid.

`EPERM`
: User does not have appropriate privileges in case of setting values.

EINTR
: A signal was caught during the `cd_mnconv()` function.

EMFILE
: `{OPEN_MAX}` file descriptors are currently open in the calling process.

ENFILE
: The system file table is full.

APPLICATION USAGE

The setting of values is restricted to a user with appropriate privileges. In case of setting file name conversion this function is intended to be used only directly after the CD-ROM has been mounted, before any access to the CD-ROM is done. If the function is applied for setting file name conversion when files or directories have already been opened, the effect of this function on these files and directories is undefined.

Fujitsu Siemens Computers Add-on Reference Manual

cd_ptrec(3)

NAME

`cd_ptrec, cd_cptrec -` read Path Table Record from CD-ROM PathTable

SYNOPSIS

```
#include <sys/cdrom.h>
int cd_ptrec(path, ptrec) char *path;
struct iso9660_ptrec *ptrec;
int cd_cptrec(path, ptrec) char *path;
char *ptrec;
```

DESCRIPTION

The function `cd_ptrec()` fills the `ptrec` structure with the contents of the Path Table Record associated with a directory which is referred to by the argument *path*. The *path* argument points to a directory within a CD-ROM file hierarchy. The declaration for `struct iso9660_ptrec` is contained in `<sys/cdrom.h>`.

The function `cd_cptrec()` copies the complete Path Table Record as recorded on the CD-ROM to the address *ptrec*. The user must allocate {`CD_MAXPTRECL`} bytes for the Path Table Record. {`CD_MAXPTRECL`} is contained in `<sys/cdrom.h>`.

RETURN VALUE

Upon successful completion, the functions return a value of zero. In case of an error, −1 is returned and `errno` is set to indicate the error.

ERRORS

The functions will fail if:

`EACCES`
: Search permission is denied for a component of the *path* prefix, or read permission is denied for the named directory.

`ENAMETOOLONG`
: The length of the *path* string exceeds {`PATH_MAX`}, or a pathname component is longer than {`NAME_MAX`} while {`_POSIX_NO_TRUNC`} is in effect.

`ENOENT`
: A component of *path* does not exist, or the *path* argument points to an empty string.

`EINVAL`
: The argument *path* points to a directory that is not within the CD-ROM file hierarchy.

`ENOTDIR`
: A component of the *path* prefix is not a directory.

`ENXIO`
: The CD-ROM is not in the drive, or a read error occurred.

`EFAULT`
: The address of *ptrec* or *path* is invalid.

`EINTR`
: A signal was caught during one of the functions.

`EMFILE`
: {`OPEN_MAX`} file descriptors are currently open in the calling process.

`ENFILE`
: The system file table is full.

Fujitsu Siemens Computers Add-on Reference Manual

cd_pvd(3)

NAME

`cd_pvd, cd_cpvd -` read Primary Volume Descriptor from CD-ROM

SYNOPSIS

```
#include <sys/cdrom.h>
int cd_pvd(path, pvd) char *path;
struct iso9660_pvd *pvd;
int cd_cpvd(path, pvd) char *path;
char *pvd;
```

DESCRIPTION

The function `cd_pvd()` fills the `pvd` structure with the contents of the Primary Volume Descriptor from the CD-ROM. The declaration for `struct iso9660_pvd` is contained in `<sys/cdrom.h>`. The *path* argument points to a pathname naming a file or directory within a CD-ROM file hierarchy, or naming a block special file for a CD-ROM file system. To execute this function successfully the user must have read or execute permission on the file/directory pointed to by *path* or an appropriate privilege.

The function `cd_cpvd()` copies the complete Primary Volume Descriptor as recorded on the CD-ROM to the address *pvd*. The user must allocate {`CD_PVDLEN`} bytes for the Primary Volume Descriptor. {`CD_PVDLEN`} is contained in `<sys/cdrom.h>`.

RETURN VALUE

Upon successful completion the functions return a value of zero. In case of an error, −1 is returned and `errno` is set to indicate the error.

ERRORS

The functions will fail is:

`EACCES`
: Search permission is denied for a component of the *path* prefix. Read permission for the file or directory pointed to by *path*, or read permission for the block special file pointed to by *path*, is denied.

`ENAMETOOLONG`
: The length of the *path* string exceeds {`PATH_MAX`}, or a pathname component is longer than {`NAME_MAX`} while {`_POSIX_NO_TRUNC`} is in effect.

`ENOENT`
: A component of *path* does not exist, or the *path* argument points to an empty string.

`EINVAL`
: The named file is a block special file and the CD-ROM is not recorded according to the ISO 9660 standard.

  The argument *path* points to a file/directory that is not within the CD-ROM file hierarchy.

`ENOTDIR`
: A component of the *path* prefix is not a directory.

`ENXIO`
: The named file is a block special file and the device associated with the special file does not exist. The CD-ROM is not in the drive, or a read error occurred.

`EFAULT`
: The address of *pvd* or *path* is invalid.

`EINTR`
: A signal was caught during one of the functions.

`EMFILE`

{OPEN_MAX} file descriptors are currently open in the calling process.

ENFILE
The system file table is full.

Fujitsu Siemens Computers Add-on Reference Manual

cd_setdevmap(3)

NAME

cd_setdevmap - set mappings of major/minor numbers

SYNOPSIS

```
#include <sys/cdrom.h>
int cd_setdevmap(path, cmd, new_major, new_minor) char *path;
int cmd;
int *new_major;
int *new_minor;
```

DESCRIPTION

This function sets or unsets (based on *cmd*) the major and minor numbers of one device file on a mounted CD-ROM. The argument *path* points to a file or directory within the CD-ROM file hierarchy.

If *cmd* is CD_SETDMAP, this function maps the *new_major* major number and the *new_minor* minor number to the device file pointed to by *path*. *new_major* specifies the new major number for the device file. *new_minor* specifies the new minor number for the device file. Any device file mapping for the device file *path* set with a previous invocation of cd_setdevmap() is overridden by this invocation of cd_setdevmap().

If *cmd* is CD_UNSETDMAP, this function unmaps the mapped major and minor numbers of the device file pointed to by *path*. The value of the recorded major number on the CD-ROM shall be returned in *new_major*. The value of the recorded minor number on the CD-ROM shall be returned in *new_minor*.

RETURN VALUE

For CD_SETDMAP, cd_setdevmap() will return one if the device file is successfully mapped (a return value of zero means no more mappings allowed).

For CD_UNSETDMAP, cd_setdevmap() will return one if the device file is successfully unmapped (a return value of zero means mapping not found).

In case of error, –1 is returned and errno is set to indicate the error.

ERRORS

EACCESS
: Search permission is denied for a component of the *path* prefix or read permission on the device file pointed to by *path* is denied.

ENAMETOOLONG
: The length of the *path* string exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect.

ENOENT
: A component of *path* does not exist or the *path* argument points to an empty string.

ENOTDIR
: A component of the *path* prefix is not a directory.

EFAULT
: The address of *path*, *new_major*, or *new_minor* is invalid.

EINVAL
: The value of *cmd* is invalid.

  The argument *path* points to a file/directory not within a CD-ROM file hierarchy.

  The file pointed to by *path* is not a device file.

EPERM
: User does not have appropriate privileges to set/unset device file major/minor values.

ENXIO
: The CD-ROM is not in the drive or a read error occurred.

EINTR
: A signal was caught during the `cd_setdevmap()` function.

EMFILE
: {`OPEN_MAX`} file descriptors are currently open in the calling process.

ENFILE
: The system file table is full.

APPLICATION USAGE

The use of `cd_setdevmap()` is restricted to a user with appropriate privileges. The maximum number of mappings is defined in `<sys/cdrom.h>`. Mappings should be established before affected device files are used. If this function is applied for device files that have already been opened, the effect of this function on these files is undefined. The device file mappings for a mounted CD-ROM are eliminated when the CD-ROM is unmounted.

Fujitsu Siemens Computers Add-on Reference Manual

cd_suf(3)

NAME

`cd_suf` - read System Use Field from a specified System Use Area

SYNOPSIS

```
#include <sys/cdrom.h>
int cd_suf(path, fsec, signature, index, buf, buflen) char *path;
int  fsec;
char signature[2];
int  index;
char *buf;
int  buflen;
```

DESCRIPTION

`cd_suf()` returns a System Use Field in the System Use Area for *path*.

path points to a file or directory within the CD-ROM file hierarchy.

fsec specifies the File Section of that file. The numbering starts with one. If *fsec* is set to −1, the System Use Area of the last File Section of that file is assumed.

signature is the 2 byte signature to look for and return from the System Use Area.

index is the occurrence of *signature* to return. If *signature* is a `NULL` pointer, return the *index* System Use Field starting from the beginning of the System Use Area. Otherwise, return the *index* occurrence of *signature*. The *index* number of the first System Use Field of any *signature* is one.

buf and *buflen* are the buffer and buffer length in which to place the System Use Field.

RETURN VALUE

`cd_suf()` will return the number of bytes placed in *buf* if successful. `cd_suf()` will return 0 if the *signature* field is not found. In case of error, −1 is returned and `errno` is set to indicate the error.

ERRORS

The `cd_suf()` function will fail if:

`EACCESS`
: Search permission is denied for a component of the *path* prefix or read permission on the file or directory pointed to by *path* is denied.

`ENAMETOOLONG`
: The length of the *path* string exceeds {`PATH_MAX`} or a pathname component is longer than {`NAME_MAX`} while {`_POSIX_NO_TRUNC`} is in effect.

`ENOENT`
: A component of *path* does not exist or the *path* argument points to an empty string.

: The File Section indicated by *fsec* has no System Use Area.

`ENOTDIR`
: A component of the *path* prefix is not a directory.

`EFAULT`
: The address of *buf*, *signature* or *path* is invalid.

`EINVAL`
: The value of *fsec*, *index* or *buflen* is invalid.

: The argument *path* points to a file/directory not within a CD-ROM file hierarchy.

`ENODEV`

The Volume containing the File Section indicated by *fsec* is not mounted.

`ENXIO`
The CD-ROM is not in the drive or a read error occurred.

`EINTR`
A signal was caught during the `cd_suf()` function.

`EMFILE`
`{OPEN_MAX}` file descriptors are currently open in the calling process.

`ENFILE`
The system file table is full.

Fujitsu Siemens Computers Add-on Reference Manual

cd_type(3)

NAME

`cd_type -` get identification of CD-ROM

SYNOPSIS

```
#include <sys/cdrom.h>
int cd_ptrec(path) char *path;
```

DESCRIPTION

This function determines the type of a CD-ROM. The argument *path* points to a pathname naming a file or directory within the CD-ROM file hierarchy, or to a pathname naming the block special file for the CD-ROM file system. The return value of `cd_tpye()` indicates the type of the CD-ROM. The definition for the return value is contained in `<sys/cdrom.h>`.

This function is intended for future expansion, when XCDR may support more than one CD-ROM type, like CD-ROM XA.

RETURN VALUE

Upon successful completion, `cd_type()` returns the following value:

`{CD_ISO9660}`
: CD-ROM is recorded according to ISO 9660.

In case of other CD-ROM types, the return value is implementation-defined. In case of an error, −1 is returned and `errno` is set to indicate the error.

ERRORS

The `cd_type()` function will fail if:

`EACCES`
: Search permission is denied for a component of the *path* prefix, or read and execute permission is denied for the named file or directory, or read permission is denied on the block special file pointed to by *path*.

`ENAMETOOLONG`
: The length of the *path* string exceeds `{PATH_MAX}`, or a pathname component is longer than `{NAME_MAX}` while `{_POSIX_NO_TRUNC}` is in effect.

`ENOENT`
: A component of *path* does not exist, or the *path* argument points to an empty string.

`EINVAL`
: The argument *path* points to a file/directory that is not within the CD-ROM file hierarchy.

`ENOTDIR`
: A component of the *path* prefix is not a directory.

`ENXIO`
: The named file is a block special file and the device associated with the special file does not exist.

: The CD-ROM is not in the drive, or a read error occurred.

`EFAULT`
: The address of *path* is invalid.

`EINTR`
: A signal was caught during the `cd_type()` function.

`EMFILE`
: `{OPEN_MAX}` file descriptors are currently open in the calling process.

`ENFILE`
: The system file table is full.

Fujitsu Siemens Computers Add-on Reference Manual

cd_xar(3)

NAME

`cd_xar, cd_cxar -` read Extended Attribute Record for CD-ROMfile/directory from CD-ROM

SYNOPSIS

```
#include <sys/cdrom.h>
int cd_xar(path, fsec, xar, applen, esclen) char *path;
int fsec;
struct iso9660_xar *xar;
int applen, esclen;
int cd_cxar(path, fsec, xar, xarlen) char *path;
int fsec;
char *xar;
int xarlen;
```

DESCRIPTION

The `cd_xar()` function fills the `xar` structure with the contents of the XAR associated with a file or directory which is referred to by the argument *path*. The argument *fsec* specifies the File Section of that file. The numbering starts with one. If *fsec* is set to −1, the XAR of the last File Section of that file is assumed. The argument *path* points to a file or directory within the CD-ROM file hierarchy. The two arguments (*applen*, *esclen*) determine how many bytes will be copied to the addresses specified in the `iso9660_xar` structure by *app_use* and *esc_seq*. The total number of logical blocks of an XAR can be obtained by the `cd_drec()` function. The Logical Block Size in bytes can be obtained by the `cd_pvd()` function. The length of the fixed part of the XAR is given by {`CD_XARFIXL`}. The declaration for `struct iso9660_xar` and the definition of {`CD_XARFIXL`} are contained in `<sys/cdrom.h>`.

The `cd_cxar()` function copies the XAR as recorded on the CD-ROM to the address *xar*. With *xarlen* set appropriately, part of the XAR or the full XAR will be read.

RETURN VALUE

The `cd_xar()` function returns the number of bytes copied for the variable part of the XAR. The `cd_cxar()` function returns the number of bytes copied. In case of an error, −1 is returned and `errno` is set to indicate the error.

ERRORS

The function will fail if:

`EACCES`
: Search permission is denied for a component of the *path* prefix, or read permission for the file or directory pointed to by *path* is denied.

`ENAMETOOLONG`
: The length of the *path* string exceeds {`PATH_MAX`}, or a pathname component is longer than {`NAME_MAX`} while {`_POSIX_NO_TRUNC`} is in effect.

`ENOENT`
: A component of *path* does not exist, or the *path* argument points to an empty string.

  The File Section indicated by *fsec* has no XAR.

`ENOTDIR`
: A component of the *path* prefix is not a directory.

`EFAULT`
: The address of *xar* or *path* is invalid.

`EINVAL`
: The value of *fsec* or *xarlen* is invalid.

  The argument *path* points to a file/directory that is not within the CD-ROM file hierarchy.

`ENODEV`
: The Volume containing the File Section indicated by *fsec* is not mounted.

`ENXIO`
: The CD-ROM is not in the drive, or a read error occurred.

`EINTR`
: A signal was caught during one of the functions.

`EMFILE`
: `{OPEN_MAX}` file descriptors are currently open in the calling process.

`ENFILE`
: The system file table is full.

SEE ALSO

`cd_drec`(3), `cd_pvd`(3).

Fujitsu Siemens Computers Add-on Reference Manual

chan_close(3-clos)
Cluster Operating System

NAME

chan_close, chan_send_pid - closing a node channel

SYNOPSIS

```
cc [ flag ... ] file ... -lmproc -lclos
#include <cluster.h>
int chan_close(int fd, long val);
int chan_send_pid(int fd, long val);
```

DESCRIPTION

A node channel refers to a FIFO that is mapped from a local node to a different node in the cluster. The FIFOs are mapped by the closmon(1M) monitor and are created in the /dev/clos/*clos_name* directory with the tag node of the relevant node. These node channels are generally opened with the open_node_channel(3-clos) function.

The chan_close() and chan_send_pid() functions supply the nodes at the opposite end of the FIFO with the *val* value and the process ID of the calling process. If a process there has itself opened a FIFO for the local node with the same ID [see set_chan_key(3-clos)], a CLOS control message is issued to this process. This is then read with the receive_chan_ctrl(3-clos) function. The *fd* argument is an open file descriptor for the channel.

chan_close()
: The chan_close() function first issues the process ID and the *val* value on the marked open node channel. It then submits the processor to give the closmon(1M) monitor the opportunity to pass on this CLOS control message. The file descriptor is then closed. The CLOS control message is delivered to the remote station as a CLOS_SENDVAL type.

chan_send_pid()
: The chan_send_pid() function issues the process ID and the *val* value on the marked open node channel. The CLOS control message is delivered to the remote station as a CLOS_RETPID type.

The CLOS control messages are assigned a high priority and have a struct ctrlresp structure. They can be read with the receive_chan_ctrl(3-clos) function. The CLOS control messages for these functions comprise the following components:

resp_val
: is the value transferred

resp_id
: is the transferred process ID of the sending process

RESULT

The value 0 is returned upon successful conclusion. Otherwise, –1 is returned and errno is set to display the error.

ERRORS

If one or more of the following conditions apply, the functions are deemed unsuccessful:

EBADF
: *fd* is not a valid file descriptor open for writing.

EBADMSG
: The message read does not comply with the format for CLOS control messages.

EINTR
: A signal was intercepted during the getmsg(2) or putmsg(2) system call.

ENOSR
: No buffer area could be assigned for the message being created because of insufficient STREAMS storage.

ENOSTR
: There is no stream associated with *fd*.

ENXIO
: A hangup was generated down-stream for the specified stream or the other end of the FIFO is closed.

FILES

`/dev/clos/`*clos_name*

SEE ALSO

`closmon`(1M), `getmsg`(2), `putmsg`(2), `receive_chan_ctrl`(3-clos), `set_chan_key`(3-clos), `open_node_channel`(3-clos), `clos`(7).

"Programmer's Guide: STREAMS".

Fujitsu Siemens Computers Add-on Reference Manual

chan_cmd(3-clos)
Cluster Operating System

NAME

chan_cmd - executing commands on the remote node of a node channel

SYNOPSIS

```
cc [ flag ... ] file ... -lmproc -lclos
#include <cluster.h>
int chan_cmd(int fd, long cid, char *cmd);
```

DESCRIPTION

A remote node refers to the node for which a node channel is mapped. The chan_cmd() function transfers a command character string to the remote node where it is executed.

Execution rights are also transferred to the target node by the chan_cmd() function where they are then evaluated. The command is executed in the $HOME directory of the user of the chan_cmd() function.

The command character string is transferred and executed asynchronously in respect of this function in two steps. Each step is acknowledged by a CLOS control message [see clos(7)]:

CLOS_ACKN
: Acknowledges the transfer of the command character string. The transfer is successful if the resp_val structure element in the CLOS_ACKN message is 0. Otherwise, the message contains an error number in accordance with the description in intro_prm2(2) .

CLOS_RETURN
: Acknowledges the execution of the command on the remote node, which sends the CLOS_RETURN message as soon as the command has terminated. The resp_val structure element contains the end status of the process as described in wstat(5) .

The CLOS control messages are assigned a high priority and can be read with the receive_chan_ctrl(3-clos) function.

The chan_cmd function simply starts the command and then returns to the caller. If the function was ended with errors (return value –1), no command is initiated and a CLOS control message is not awaited.

ARGUMENTS

fd This is the file descriptor of the node channel for the remote node.

cid This is the command ID for the caller. Since the commands are executed asynchronously in respect of the active process, a process can start several commands for execution at the same time. In order to distinguish between these commands, an ID can be assigned by the process which is then returned to the process as an acknowledgement in the resp_id structure element of the CLOS control message.

cmd This is the command character string that is to be executed on the remote node. The character string must be terminated by \0 (zero).

RESULT

When the command has been issued successfully, the value 0 is returned. Otherwise, –1 is returned and errno is set to display the error. A CLOS control message is not awaited in this case.

ERRORS

If one or more of the following conditions apply, the function is deemed unsuccessful:

EBADF
: *fd* is not a valid file descriptor open for writing.

EINVAL
: The *key* argument has an invalid 0 value.

EINTR
: A signal was intercepted during the putmsg(2) system call.

ENOSR
: No buffer area could be assigned for the message being created because of insufficient STREAMS storage.

ENOSTR
: There is no stream associated with *fd*.

ENXIO
: A hangup was generated down-stream for the specified stream or the other end of the FIFO is closed.

SEE ALSO

putmsg(2), receive_chan_ctrl(3-clos), wstat(5), clos(7).

"Programmer's Guide: STREAMS".

Fujitsu Siemens Computers Add-on Reference Manual

chan_is_reachable(3-clos)

NAME

`chan_is_reachable -` checking the accessibility of a node channel

SYNOPSIS

`cc` [*flag ...*] *file ...* `–lmproc –lclos`

```
#include <cluster.h>
#include <fcntl.h>
int chan_is_reachable(char *tagp);
```

DESCRIPTION

The `chan_is_reachable()` function only checks one node channel with the *tagp* tag name. If communication with a node is possible, the value 0 is returned. If it is not possible, –1 is returned and `errno` specifies the reason.

The file descriptor is returned upon successful conclusion. Otherwise, –1 is returned and `errno` is set to display the error.

ERRORS

If one or more of the following conditions apply, the function is deemed unsuccessful:

`ENODEV`
: The node assigned to the *tagp* tag name is not currently integrated in the cluster.

`ENOENT`
: The node assigned to the *tagp* tag name is not currently configured in the cluster.

`ENOTDIR`
: The node assigned to the *tagp* tag name is not currently configured in the cluster.

`EINVAL`
: The value of the *mode* argument is invalid.

`EACCES`
: The *mode* permission for an existing FIFO is denied.

`EINTR`
: A signal was intercepted during the `open(2)` system call.

`EIO` A connection cleardown (hangup) or an error occurred while opening the FIFO.

`EMFILE`
: The process has too many open files [see `getrlimit(2)`].

`ENFILE`
: The system file table is full.

`ENOSR`
: A stream cannot be assigned.

`ENOMEM`
: The system cannot assign any resources.

FILES

`/dev/clos/`*clos_name*

SEE ALSO

`getrlimit(2)`, `open(2)`, `fcntl(5)`, `clos(7)`.

"Programmer's Guide: STREAMS".

Fujitsu Siemens Computers Add-on Reference Manual

clos_daemon(3-clos)
Cluster Operating System

NAME

clos_daemon:
daemon_start, get_pid_lock, test_pid_lock - functions for supporting the CLOS daemon processes

SYNOPSIS

```
cc [ flag ... ] file ... -lmproc -lclos
#include <cluster.h>
int daemon_start(char *path);
int get_pid_lock(char *path);
int test_pid_lock(char *path);
```

DESCRIPTION

These functions are auxiliary routines that facilitate the development of CLOS daemon processes [see clos(7)].

The daemon_start() function creates a daemon process from the current process. This process can be characterized by such features that it

- does not have a controlling TTY
- has not opened a file descriptor, and also not standard input, standard output and standard error output
- does not belong to any process group
- is not a "Process Group Leader", i.e. it ignores the SIGCHLD signals of its child processes [see signal(5)]
- has deleted the file mode creation mask [see umask(2)]

As its working directory, the function uses the directory that was transferred as the *path* parameter. When a daemon process has been started successfully, the value 0 is returned.

The get_pid_lock() function creates the lock file with the pathname from the *path* argument. The process number is entered in the file and the entire file is locked with the lockf(3C) function. When the file has been created and locked successfully, the file descriptor of the opened file with the *path* name is returned. If, on the other hand, the file could not be created or if the file was already locked, -1 is returned as a value and errno is set.

The test_pid_lock() function checks the file with the *path* pathname. The check covers whether the file already exists and is locked. The global errno variable is *always* set by this function. The return value of the routine is:

0	If the file does not exist, errno is set to 0.
-1	If the file does exist but is not locked (EEXISTS) or if the access rights for the process are set such that access is denied (EACCES).
PID	A positive figure identifies the process number currently locked by the *path* file. The errno variable is EBUSY in this case.

RESULT

When the function has been executed successfully, the value 0 or a positive value is returned. In the case of errors, -1 is returned and errno is set to display the error.

ERRORS

ENOTDIR

A component in the *path* pathname is not a directory.

`EAGAIN`
The file with the *path* pathname is locked by another process.

`EACCES`
Permission to create the file with the *path* pathname is not sufficient for the user.

`EEXISTS`
The file with the *path* pathname exists but is not locked.

`EBUSY`
The file with the *path* pathname exists and is locked.

`EINTR`
A signal was intercepted during the `open(2)` system call.

`EMFILE`
The process has too many open files [see `getrlimit(2)`].

`ENFILE`
The system file table is full.

`ENOMEM`
The system cannot assign any resources.

SEE ALSO

`closmon(1M)`, `getrlimit(2)`, `umask(2)`, `signal(5)`, `clos(7)`.

Fujitsu Siemens Computers Add-on Reference Manual

get_clos_name(3-clos)
Cluster Operating System

NAME

get_clos_name, CLOS_name - determining the selected cluster names

SYNOPSIS

```
cc [ flag ... ] file ... -lmproc -lclos
#include <cluster.h>
int get_clos_name(char *namep);
extern char *CLOS_name;
```

DESCRIPTION

As described in clos(7), every cluster has a name. This name is assigned to the local node using the clos_adm(1M) command and the node is assigned to a cluster with the node_adm(1M) command.

In the case of all CLOS commands, the name of the cluster can be set using

- the -C option
- the $CLOSNAME environment variable [see environ(5)]
- or the default setting in the /var/clos/.CLOSdefault file

The get_clos_name() function determines the name of the selected cluster. This function requests memory for names with malloc(3C) and places the address in the CLOS_name variable. The CLOSNAME environment variable is set using the putenv(3C) function.

The *namep* argument is a pointer to a character string. If this pointer is not NULL, the character string is copied to the CLOS_name address and the function is terminated. If the *namep* argument is NULL, the CLOSNAME environment variable is copied to the CLOS_name address. If this variable is also not set, the /var/clos/.CLOSdefault file is read and the value entered here is copied to the CLOS_name global variable. The CLOSNAME environment variable is then set using the putenv(3C) function.

RESULT

The get_clos_name() function enters the cluster name established in the CLOS_name global variable, sets the CLOSNAME environment variable and terminates with the return value 0. If an error situation has occurred, this function returns the value -1 and sets the errno global variable to display the error.

ERRORS

If one or more of the following conditions apply, the function is deemed unsuccessful:

EACCES
: The process is not authorized to read the /var/clos/.CLOSdefault file.

ENODATA
: The /var/clos/.CLOSdefault file does not contain any cluster names.

ENOENT
: The /var/clos/.CLOSdefault file does not exist.

EINTR
: A signal was intercepted during a system call.

FILES

/var/clos/.CLOS_default

SEE ALSO

clos_adm(1M), node_adm(1M), malloc(3C), putenv(3C), environ(5), clos(7).

Fujitsu Siemens Computers Add-on Reference Manual

get_tagnames(3-clos)
Cluster Operating System

NAME

`get_tagnames, get_this_tag, hosttotag, tagtohost -` establishing the tag names of the active nodes

SYNOPSIS

```
cc [ flag ... ] file ... -lmproc -lclos
#include <cluster.h>
int get_tagnames(char *tags[]);
int get_this_tag(char *tagp);
char *hosttotag(char *hostname);
char *tagtohost(char *tagname);
```

DESCRIPTION

Tag names are the names of the nodes as specified by the cluster administration. They are neither the Internet names nor the node names of the systems. To enable communication, the `Closmon` monitor [see `closmon`(1M)] creates FIFOs in the `/dev/clos/`*clos_name* directory. These FIFOs bear the tag name of the node for which a channel has been mapped. If the file is not a FIFO but rather a normal file, the node is not integrated but is configured in the cluster.

`get_tagnames()`

Using the `get_tagnames()` function, a `CLOS` subsystem can determine the tag names of all the configured nodes in the cluster. No tag name is determined if the node on which this routine is used is not integrated in the cluster.

The *tags* argument is an array of pointers to character strings. The dimension of the array is `MAX_NODES` and must have been created by the caller. The `get_tagnames()` function requests memory for every tag name with `malloc`(3C) and ends the name with the character \0.

`get_this_tag()`

The `get_this_tag()` function determines the tag name of the local node and copies it as a string to the address specified in the *tagp* parameter. The function obtains this tag name from the `/var/clos/`*clos_name*`/nodes/this/tag` file. In the event of an error, the routine returns -1 and sets `errno` accordingly.

`hosttotag()`

The `hosttotag()` function determines the tag name of the node whose network name was transferred as the *hostname* argument. The result is a pointer to the tag name in a static buffer. The buffer is overwritten the next time this function is called. The function reads the local `/etc/saf/`*clos_name*`/_pmtab` file and searches through it for the host name. If the host name is not found, the value `NULL` is returned. If the attempt to open [see `fopen`(3S)] the file is rejected, `NULL` is likewise returned. The host name can be the official host name, an alias name or the Internet name in dot notation. The `localhost` and `127.0.0.1` names are also permitted for the local nodes.

`tagtohost()`

The `tagtohost()` function determines the host name of the node whose tag name was transferred as the *tagname* argument. The result is a pointer to the host name in a static buffer. The buffer is overwritten the next time this function is called. The function reads the `/etc/saf/`*clos_name*`/_pmtab` local file and searches through it for the tag name. If the tag name is not found, the value `NULL` is returned. If the attempt to open [see `fopen`(3S)] the file is rejected, `NULL` is likewise returned. The host name is output is accordance with the entry in the `/etc/saf/`*clos_name*`/_pmtab` file.

RESULT

The `get_tagnames` function enters the pointer to the names in the *tags* array beginning with the index 0. The names are arranged in alphabetical order. The indices in which no names are entered remain unchanged. The return value of the routine specifies the number of tag names entered. If the local node is not integrated, the value is 0. If the `/dev/clos/`*clos_name* directory is not available and the local node is not configured for the cluster, the return value is −1.

The `get_this_tag()` function returns the value 0 after the tag name has been copied. In the event of an error, −1 is returned and `errno` is set.

The `tagtohost()` and `hosttotag()` functions return a pointer to a static buffer that is overwritten the next time either of the two functions is called. The names are therefore only valid until the next program call.

ERRORS

If one or more of the following conditions apply, the function is deemed unsuccessful:

`EACCES`
: The process is not authorized to read the `/var/clos/`*clos_name*`/nodes/this/tag` file.

`ENODEV`
: The local node is not configured in the cluster.

`ENOENT`
: The `/var/clos/`*clos_name*`/nodes/this/tag` file does not exist. The local node is currently not `INTEGRATED`.

`EINTR`
: A signal was intercepted during a system call.

FILES

`/dev/clos/`*clos_name*

`/etc/saf/`*clos_name*`/_pmtab`

`/var/clos/`*clos_name*`/nodes/this/tag`

SEE ALSO

`closmon`(1M), `malloc`(3C), `fopen`(3S), `clos`(7).

Fujitsu Siemens Computers Add-on Reference Manual

mon_cmd_rq(3-clos)
Cluster Operating System

NAME

mon_cmd_rq - requesting the execution of shell commands

SYNOPSIS

```
cc [ flag ... ] file ... -lmproc -lclos
#include <cluster.h>
int mon_cmd_rq(char *tags[], char *cmd);
```

DESCRIPTION

The mon_cmd_rq() function issues a request to the local Closmon monitor [see closmon(1M)] indicating that the *cmd* command is to be executed on every node in the *tags*[] field. The monitor sends the command string to each of the named nodes if their status is INTEGRATED [see clos(7)]. An sh(1) shell is started on every addressed node. The user name is also passed to the nodes where it is used to set the environment for the shell from the passwd(4) file. The shell is executed, in particular, in the $HOME directory with the $HOME, $PATH and $USER environment variables [see environ(5)].

ARGUMENTS

tags[]
: This is a field containing pointers to the tag names of the nodes on which the command is to be executed. The field is completely filled beginning with the 0 index and ending with a NULL pointer. The maximum index is MAX_NODES. If more than MAX_NODES pointers are entered in the field, E2BIG is returned. The get_tagnames(3-clos) function can be used to fill the field.

  If no tag name is entered in the *tags*[] field (0 index contains NULL), the command is executed on *all* integrated nodes.

cmd
: This is a pointer to a character string that specifies the command to be executed.

RESULT

The value 0 is returned after the function is successfully executed. If an error occurs, a value other than 0 is returned and errno is set to display the error. The following return values are possible:

1
: The field in the *tags* argument is larger than MAX_NODES. errno is set to E2BIG.

2
: The write(2) system call for writing to the /etc/saf/*clos_name*/_pmpipe file failed. errno is set to display the error.

-1
: The open(2) system call for the /etc/saf/*clos_name*/_pmpipe command FIFO of the Closmon monitor failed. errno is set to display the error.

ERRORS

E2BIG
: The *tags* argument contains more than MAX_NODES pointers.

ENODEV
: The local node is not currently integrated in the cluster.

ENOENT
: The local node is not currently configured in the cluster.

EINVAL
: The *cmd* argument points to an invalid address.

EACCES
: Permission to transmit the job for the _pmpipe FIFO of the Closmon monitor is denied.

EINTR

A signal was intercepted during the write(2) or open(2) system call.

EIO A connection cleardown (hangup) or an error occurred when opening the _pmpipe FIFO.

EAGAIN
There is insufficient buffer capacity available for writing to the _pmpipe FIFO because another process has just written data to the buffer and the Closmon monitor has not yet retrieved it.

EMFILE
The process has too many open files [see getrlimit(2)].

ENFILE
The system file table is full.

ENOSR
A stream cannot be allocated.

ENOMEM
The system cannot assign any resources.

FILES

/etc/passwd

/etc/saf/*clos_name*/_pmpipe

SEE ALSO

sh(1), closmon(1M), getrlimit(2), open(2), write(2), get_tagnames(3-clos), passwd(4), environ(5), clos(7).

Fujitsu Siemens Computers Add-on Reference Manual

mon_send_rq(3-clos)
Cluster Operating System

NAME

mon_send_rq - requesting the distribution of files

SYNOPSIS

```
cc [ flag ... ] file ... -lmproc -lclos
#include <cluster.h>
int mon_send_rq(char *tags[], char *path);
```

DESCRIPTION

The mon_send_rq() functions issues a request to the local Closmon monitor [see closmon(1M)] indicating that the *path* file is to be copied on every node in the *tags*[] field. The monitor sends the file name as a string to each of the nodes identified if their status is INTEGRATED [see clos(7)]. A file bearing this pathname is created on every addressed node. The user name and file attributes are also transmitted to the nodes where they are retained. If the user is not known, the superuser becomes the owner of the file.

ARGUMENTS

tags[]
: This is a field containing pointers to the tag names of the nodes on which the command is to be executed. The field is completely filled beginning with the 0 index and ending with a NULL pointer. The maximum index is MAX_NODES. If more than MAX_NODES pointers are entered in the field, E2BIG is returned. The get_tagnames(3-clos) function can be used to fill the field.

  If no tag name is entered in the *tags*[] field (0 index contains NULL), the file is distributed to *all* integrated nodes.

path This is a pointer to a character string that specifies the *complete* pathname of the file to be distributed.

RESULT

The value 0 is returned after the function is successfully executed. If an error occurs, a value other than 0 is returned and errno is set to display the error. The following return values are possible:

1 The field in the *tags* argument is greater than MAX_NODES. errno is set to E2BIG.

2 The write(2) system call for writing to the /etc/saf/*clos_name*/_pmpipe file failed. errno is set to display the error.

-1 The open(2) system call for the /etc/saf/*clos_name*/_pmpipe command FIFO of the Closmon monitor failed. errno is set to display the error.

ERRORS

E2BIG
: The *tags* argument contains more than MAX_NODES pointers.

ENODEV
: The local node is not currently integrated in the cluster.

ENOENT
: The local node is not currently configured in the cluster.

EINVAL
: The *cmd* argument points to an invalid address.

EACCES
: Permission to transmit the request for the _pmpipe FIFO of the Closmon monitor is denied.

EINTR
: A signal was intercepted during the write(2) or open(2) system call.

`EIO` A connection cleardown (hangup) or an error occurred when opening the `_pmpipe` FIFO.

`EAGAIN`
There is insufficient buffer capacity for writing to the `_pmpipe` FIFO because another process has just written data to the buffer and the `Closmon` monitor has not yet retrieved it.

`EMFILE`
The process has too many open files [see `getrlimit(2)`].

`ENFILE`
The system file table is full.

`ENOSR`
A stream cannot be allocated.

`ENOMEM`
The system cannot assign any resources.

FILES

`/etc/saf/`*clos_name*`/_pmpipe`

SEE ALSO

`closmon(1M)`, `getrlimit(2)`, `open(2)`, `write(2)`, `get_tagnames(3-clos)`, `clos(7)`.

Fujitsu Siemens Computers Add-on Reference Manual

mon_signal(3-clos)
Cluster Operating System

NAME

`mon_signal` - requesting the signaling of node status changes

SYNOPSIS

`cc` [*flag* ...] *file* ... `–lmproc –lclos`

```
#include <cluster.h>
int mon_signal(int signo);
```

DESCRIPTION

The `mon_signal()` function issues a request to the local `Closmon` monitor [see `closmon(1M)`] indicating that every time a connection is set up to a node by the `Closmon` monitor, this must be reported to the requesting process using a signal. If an existing connection is terminated, this is not reported since an EOF is sent on the appropriately opened channel in this case (a `read(2)` call is ended with the return value 0).

The signal number is transmitted to the monitor with the *signo* argument. The person using this function should then handle this signal accordingly [see `signal(2)` or `sigaction(2)`]. Permitted signal numbers are described in `signal(5)` .

The changes in node status that need to be monitored refer to changes in the communication status. A signal is sent if a node

- is added to the cluster group
- changes from the status `NOT_AVAIL` to `INTEGRATED`
- changes from the status `DISABLED` to `INTEGRATED`

After the signal has been supplied, the request is deemed to have been satisfied. Any subsequent changes are not signaled. If you wish the signaling process to continue, you need to issue a new `mon_signal()` call.

RESULT

The value 0 is returned after this function has been successfully executed. Otherwise, –1 is returned and `errno` is set to display the error.

ERRORS

`ENODEV`
: The local node is not currently integrated in the cluster.

`ENOENT`
: The local node is not currently configured in the cluster.

`EINVAL`
: The value of the *signo* argument is invalid.

`EACCES`
: Permission to transmit the request for the `_pmpipe` FIFO of the `Closmon` monitor is denied.

`EINTR`
: A signal was intercepted during the `write(2)` or `open(2)` system call.

`EIO` A connection cleardown (hangup) or an error occurred when opening the `_pmpipe` FIFO.

`EMFILE`
: The process has too many open files [see `getrlimit(2)`].

`ENFILE`
: The system file table is full.

`ENOSR`
: A stream cannot be allocated.

ENOMEM
The system cannot assign any resources.

FILES

/etc/saf/*clos_name*/_pmpipe

SEE ALSO

closmon(1M) open(2), read(2), sigaction(2), signal(2), write(2), signal(5), clos(7).

Fujitsu Siemens Computers Add-on Reference Manual

open_node_channel(3-clos)

NAME

`open_node_channel` - opening a node channel

SYNOPSIS

`cc` [*flag* ...] *file* ... `–lmproc –lclos`

```
#include <cluster.h>
#include <fcntl.h>
int open_node_channel(char *tagp, mode_t mode);
```

DESCRIPTION

The `open_node_channel()` function opens a node channel for the node indicated by the *tagp* argument. In this case, *tagp* is a pointer to the tag name of the node. This can be determined using `get_tagnames(3-clos)` . Tag names are the names of the nodes as specified by the cluster administration [see `clos(7)`]. When opening a node channel, a check is made to determine whether the selected node is configured and whether communication with this node is possible. Following a successful `open(2)` call to a channel FIFO, the function returns the file descriptor to the FIFO for the node.

The read and write access rights when opening a node channel are specified in the *mode* argument. The following values are permitted:

`O_RDONLY`
: Read only

`O_WRONLY`
: Write only

`O_RDWR`
: Read and write

`O_NDELAY` or `O_NONBLOCK`
: No block

These arguments are used as described for `open(2)` . However, all other arguments permitted for `open(2)` are not permitted here.

ERRORS

`ENODEV`
: The node assigned to the *tagp* tag name is not currently integrated in the cluster.

`ENOENT`
: The node assigned to the *tagp* tag name is not currently configured in the cluster.

`ENOTDIR`
: The node assigned to the *tagp* tag name is not currently configured in the cluster.

`EINVAL`
: The value of the *mode* argument is invalid.

`EACCES`
: The *mode* permission for an existing FIFO is denied.

`EINTR`
: A signal was intercepted during the `open(2)` system call.

`EIO` When opening the FIFO, a connection cleardown (hangup) or an error occurred.

`EMFILE`
: The process has too many open files [see `getrlimit(2)`].

`ENFILE`
: The system file table is full.

ENOSR
: A stream cannot be assigned.

ENOMEM
: The system cannot assign any resources.

FILES

/dev/clos/*clos_name*

SEE ALSO

getrlimit(2), open(2), fcntl(5), clos(7).

"Programmer's Guide: STREAMS".

Fujitsu Siemens Computers Add-on Reference Manual

receive_chan_ctrl(3-clos)

NAME

receive_chan_ctrl - receiving CLOS control messages

SYNOPSIS

cc [*flag ...*] *file ...* –lmproc –lclos

```
#include <cluster.h>
int receive_chan_ctrl(int fd, struct ctrlresp *rp);
```

DESCRIPTION

A user process can execute control functions on the node channels [see clos(7)]. The CLOS communication system then responds to these functions with CLOS control messages. These CLOS control messages have a high priority and, apart from a type, contain a data area for displaying the results of previous control commands. This data area is the struct ctrlresp structure with the following elements:

```
resp_id;
resp_val;
```

A distinction is made between the following CLOS control messages:

CLOS_OPEN
: This is the response to a node channel in which the set_chan_ctrl() control function is used to signal the status for the remote station. resp_val specifies the number of open channels with the same flag on the remote channel. The resp_id is 0.

CLOS_CLOSE
: This is the response to a node channel in which the set_chan_ctrl() control function is used to signal the status for the remote station. The resp_val and resp_id structure elements are 0. This implies that a channel with this flag is no longer open on the remote station.

CLOS_ACKN
: A command for the node on the remote station was issued on the node channel using the chan_cmd() function. This message signals that the CLOS communication system has accepted the command on the remote node. The resp_val structure element contains the error number if transmission of the command failed. If the value is 0, the command is executed without transmission errors. The resp_id structure element contains the command ID set by the requester.

CLOS_RETURN
: A command for the node on the remote station was issued on the node channel using the chan_cmd() function. The CLOS communication system uses this message to signal that it has received the command on the remote node. The resp_val structure element contains the end status of the command as described in wstat(5) . The resp_id structure element contains the command ID set by the requester.

CLOS control messages are sent *spontaneously* if

- a channel with the same ID is open on the remote station
- a channel with the same ID was closed on the remote station
- a command reached the remote station
- a command was ended on the remote node

The user can read the CLOS control messages with the receive_chan_ctrl() function.

RESULT

If a CLOS control message exists, the struct respctrl structure is filled and the type of CLOScontrol message is returned. If the *fd* file descriptor was opened in non-blocking mode [see O_NDELAY or O_NONBLOCK in fcntl(5)], the function returns immediately. If the return value is 0, there were no CLOS

control messages and `resp_val` is –1. If the return value is positive, this means that a `CLOS` control message has been read and the `struct respctrl` structure is filled with valid values. If there is no `CLOS` control message and if neither `O_NDELAY` nor `O_NONBLOCK` is set, the function is blocked. If –1 is returned, an error is indicated and `errno` displays the error. The *fd* argument is an open file descriptor on the channel.

ERRORS

If one or more of the following conditions apply, the functions are deemed unsuccessful:

`EBADF`
: *fd* is not a valid file descriptor open for writing.

`EBADMSG`
: The message read does not comply with the `CLOS` control message format.

`EINTR`
: A signal was intercepted during the `getmsg(2)` or `putmsg(2)` call.

`ENOSTR`
: There is no stream associated with *fd*.

`ENXIO`
: A hangup was generated down-stream for the specified stream or the other end of the FIFO is closed.

FILES

`/dev/clos/`*clos_name*

SEE ALSO

`getmsg(2)`, `putmsg(2)`, `chan_cmd(3-clos)`, `set_chan_ctrl(3-clos)`, `fcntl(5)`, `wstat(5)`, `clos(7)`.

"Programmer's Guide: STREAMS".

Fujitsu Siemens Computers Add-on Reference Manual

send_chan_ctrl(3-clos)
Cluster Operating System

NAME

`send_chan_ctrl:`
`set_chan_key, set_chan_ctrl -` controlling the node channel properties

SYNOPSIS

`cc [` *flag ...* `]` *file ...* `–lmproc –lclos`

```
#include <cluster.h>
int set_chan_key(int fd, long key);
int set_chan_ctrl(int fd, long key);
```

DESCRIPTION

A node channel refers to a FIFO that is mapped from the local node to another node in the cluster. These FIFOs are mapped by the `Closmon` monitor [see `closmon`(1M)] and are created in the `/dev/clos/`*clos_name* directory with the tag name of the corresponding node.

Using the `send_chan_ctrl` functions, a control command is issued on an open node channel which is used to set the channel properties. The *fd* argument is an open file descriptor on the channel.

`set_chan_key()`
The `set_chan_key()` function marks an open channel with the *key* flag.

`set_chan_ctrl()`
The `set_chan_ctrl()` function also marks a channel and awaits receipt of the following types of `CLOS` control messages on the channel:

`CLOS_OPEN`
This signals that a channel with the same flag was opened on the remote station.

`CLOS_CLOSE`
This signals that a channel with the same flag was closed on the remote station.

The `CLOS` control messages have a high priority and contain a data area for displaying the number of channels with this flag that are currently open. These channels are read using the `receive_chan_ctrl`(3-clos) function.

The `CLOS` control messages are sent spontaneously in the following cases:

- immediately after this property has been enabled
- if (another) channel with this ID is opened on the remote station
- if a channel with this ID is closed on the remote station

RESULT

Only the channel properties are set in the `Closmon` monitors using the routines. A command is not transmitted to any other node. After the command has been executed successfully, 0 is returned. Otherwise, –1 is returned and `errno` is set to display the error.

ERRORS

If one or more of the following conditions apply, the functions are deemed unsuccessful:

`EBADF`
fd is not a valid file descriptor open for writing.

`EBADMSG`
The message read does not comply with the `CLOS` control message format.

`EINVAL`
The *key* argument has the invalid value 0.

EINTR
A signal was intercepted during the getmsg(2) or putmsg(2) call.

ENOSR
No buffer area could be assigned for the message being created because of insufficient STREAMS storage.

ENOSTR
There is no stream associated with *fd*.

ENXIO
A hangup was generated down-stream for the specified stream or the other end of the FIFO is closed.

FILES

/dev/clos/*clos_name*

SEE ALSO

closmon(1M), getmsg(2), putmsg(2), receive_chan_ctrl(3-clos), clos(7).
"Programmer's Guide: STREAMS".

Thread Functions(3)

ctime_r(3)

NAME

ctime_r, asctime_r, localtime_r - thread-safe converttime and date to string

SYNOPSIS

```
#include <time.h>
char *ctime_r(const time_t *clock, char *buf);
char *asctime_r(const struct tm *tm, char *buf);
struct tm *localtime_r(const time_t *clock, struct tm *result);
```

DESCRIPTION

ctime_r(), asctime_r() and localtime_r() convert the time value pointed to by *clock* and *tm* resp. in exactly the same form as ctime(), asctime() and localtime().

ctime_r() and asctime_r() put the result into the array pointed to by *buf* (which contains at least 26 bytes). localtime_r() puts the result into the structure pointed to by *result*.

Unlike ctime() and localtime(), the thread-safe functions ctime_r() and localtime_r() are not required to set tzname.

RETURN VALUE

Upon successful completion ctime_r() and asctime_r() return a pointer to the string pointed to by *buf*, localtime_r() returns a pointer to the structure pointed to by *result*. When an error is encountered, a NULL pointer is returned.

ERRORS

No errors are defined.

SEE ALSO

asctime(3C), ctime(3C), localtime(3C), time(5).

Fujitsu Siemens Computers Add-on Reference Manual

getgrgid_r(3)

NAME

`getgrgid_r` - thread-safe get group database entry for a group ID

SYNOPSIS

```
#include <grp.h>
int getgrgid_r(gid_t gid, struct group *grp, char *buffer,
        size_t bufsize, struct group **result);
```

DESCRIPTION

The `getgrgid_r()` function updates the group structure pointed to by *grp* and stores a pointer to that structure at the location pointed to by *result*. The structure contains an entry from the group database with a matching *gid*. Storage referenced by the group structure is allocated from the memory provided with the *buffer* parameter, which is *bufsize* characters in size. A `NULL` pointer is returned at the location pointed to by result on error or if the requested entry is not found.

Applications wishing to check for error situations should set `errno` to 0 before calling `getgrgid_r()`. If `errno` is set to non-zero on return, an error occurred.

RETURN VALUE

If successful, the `getgrgid_r()` function returns zero. Otherwise, an error number is returned to indicate the error.

ERRORS

The `getgrgid_r()` function may fail if:

`ERANGE`
: Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be referenced by the resulting group structure.

SEE ALSO

`getgrgid`(3C), `grp`(5).

Fujitsu Siemens Computers Add-on Reference Manual

gethostent_r(3)

NAME

gethostent_r, sethostent_r, endhostent_r - thread-safeget host name

SYNOPSIS

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
struct hostent *gethostent_r(FILE **fp);

sethostent_r(int stayopen, FILE **fp);

endhostent_r(FILE **fp);
```

DESCRIPTION

gethostent_r(), sethostent_r(), endhostent_r() work exactly as gethostent(), sethostent(), endhostent(). To provide thread-safety, gethostent_r(), sethostent_r(), endhostent_r() expect to be passed additionally the address *fp* of a file pointer as the last argument. The file pointer has to be NULL when used the first time. They will use this argument to keep track of the current file position being worked with.

RETURN VALUES

gethostent_r(), sethostent_r(), endhostent_r() return the same values as gethostent(), sethostent(), endhostent().

EXAMPLES

```
FILE *fp = NULL;
struct hostent *hp;
hp = gethostent_r(&fp);
```

SEE ALSO

endhostent(3N), gethostent(3N), sethostent(3N), netdb(5), socket(5), stdio(5), types(5).

Fujitsu Siemens Computers Add-on Reference Manual

getpwnam_r(3)

NAME

`getpwnam_r, getpwuid_r -` thread-safe search user database fora user id or name

SYNOPSIS

```
#include <sys/types.h>
#include <pwd.h>
int getpwnam_r(const char *nam, struct passwd *pwd, char *buffer,
size_t bufsize, struct passwd **result);
int getpwuid_r(uid_t *nam, struct passwd *pwd, char *buffer,
size_t bufsize, struct passwd **result);
```

DESCRIPTION

The `getpwnam_r()` and `getpwuid_r()` functions update the passwd structure pointed to by *pwd* and store a pointer to that structure at the location pointed to by *result*.

The structure will contain an entry from the user database with a matching *name* or *uid*. Storage referenced by the structure is allocated from the memory provided with the *buffer* parameter, which is *bufsize* characters in size.

A `NULL` pointer is returned at the location pointed to by *result* on error or if the requested entry is not found.

RETURN VALUE

If successful, the `getpwnam_r()` and `getpwuid_r()` functions return zero. Otherwise, an error number is returned to indicate the error.

ERRORS

The `getpwnam_r()` and `getpwuid_r()` functions may fail if:

`ERANGE`
: Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be referenced by the resulting group structure.

SEE ALSO

`getpwnam`(3C), `getpwuid`(3C), `pwd`(5), `types`(5).

Fujitsu Siemens Computers Add-on Reference Manual

rand_r(3)

NAME

`rand_r` - thread-safe pseudo-random number generator

SYNOPSIS

```
#include <stdlib.h>
int rand_r(unsigned int *seed);
```

DESCRIPTION

The `rand_r()` function is the thread-safe version of `rand()`.

The `rand_r()` function computes a sequence of pseudo-random integers in the range 0 to `RAND_MAX`. (The value of the `RAND_MAX` macro will be at least 32767.)

If `rand_r()` is called with the same initial value for the object pointed to by seed and that object is not modified between successive returns and calls to `rand_r()`, the same sequence shall be generated.

RETURN VALUE

The `rand_r()` function returns a pseudo-random integer.

ERRORS

No errors are defined.

SEE ALSO

`rand`(3C), `stdlib`(5).

Fujitsu Siemens Computers Add-on Reference Manual

readdir_r(3)

NAME

`readdir_r` - thread-safe read directory

SYNOPSIS

```
#include <sys/types.h>
#include <dirent.h>
int readdir_r(DIR *dirp, struct dirent *entry,
      struct dirent **result);
```

DESCRIPTION

The `readdir_r()` is the tread-safe version of the `readdir()` function.

The `readdir_r()` function initialises the `dirent` structure referenced by *entry* to represent the directory entry at the current position in the directory stream referred to by *dirp*, store a pointer to this structure at the location referenced by *result*, and positions the directory stream at the next entry.

The storage pointed to by *entry* will be large enough for a `dirent` with an array of `char d_name` member containing at least `NAME_MAX` plus one elements.

On successful return, the pointer returned at result will have the same value as the argument entry. Upon reaching the end of the directory stream, this pointer will have the value `NULL`.

The `readdir_r()` function will not return directory entries containing empty names. It is unspecified whether entries are returned for dot or dot-dot.

If a file is removed from or added to the directory after the most recent call to `opendir()` or `rewinddir()`, whether a subsequent call to `readdir_r()` returns an entry to that file is unspecified.

The `readdir_r()` function may buffer several directory entries per actual read operation; the `readdir_r()` function marks for update the `st_atime` field of the directory each time the directory is actually read.

RETURN VALUE

If successful, the `readdir_r()` function returns zero. Otherwise, an error number is returned to indicate the error.

ERRORS

The `readdir_r()` function may fail if:

`EBADF`
: The *dirp* argument does not refer to an open directory stream.

SEE ALSO

`readdir(3C)`, `dirent(4)`, `types(5)`.

strtok_r(3)

NAME

`strtok_r` - split string into tokens

SYNOPSIS

```
#include <string.h>
char *strtok_r(char *s, const char *sep, char
**lasts);
```

DESCRIPTION

The `strtok_r()` function is the thread-safe version of `strtok()`.

The function `strtok_r()` considers the null-terminated string *s* as a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *sep*. The argument *lasts* points to a user-provided pointer which points to stored information necessary for `strtok_r()` to continue scanning the same string.

In the first call to `strtok_r()`, *s* points to a null-terminated string, *sep* to a null-terminated string of separator characters and the value pointed to by *lasts* is ignored. The function `strtok_r()` returns a pointer to the first character of the first token, writes a null character into *s* immediately following the returned token, and updates the pointer to which *lasts* points.

In subsequent calls, *s* is a `NULL` pointer and *lasts* will be unchanged from the previous call so that subsequent calls will move through the string *s*, returning successive tokens until no tokens remain. The separator string *sep* may be different from call to call. When no token remains in *s*, a `NULL` pointer is returned.

RETURN VALUE

The function `strtok_r()` returns a pointer to the token found, or a `NULL` pointer when no token is found.

ERRORS

No errors are defined.

SEE ALSO

`strtok`(3C), `string`(5).

Thread Functions(3-thr)

Introduction

thr_intro(3-thr)

NAME

`thr_intro` - introduction to DCE Threads in Reliant UNIX 5.44 and 5.45

DESCRIPTION

About multi-threaded programming

The choice of multi-threading is really a question of specific application design. Multi-threaded programming, in general, requires a number of practices that are likely to be unfamiliar and unintuitive to many programmers, and errors arising from failure to follow these practices can be obscure, infrequent and difficult to reproduce. There are many error possibilities in a multi-threaded program that can result in all kinds of deadlocks, race conditions, and even data corruption (see the section **Thread-safeness and reentrance**).

Cancels and signals introduce a number of specific semantic issues that applications must be aware of when programming in a multi-threaded environment.

Correct use of threads mechanisms means following a set of general rules designed to avoid errors that may or may not occur in specific cases. Rules like this are, in general, not enforced by the thread programming mechanisms, and failure to follow them will not always result in program failures (see the section **Synchronization strategies and deadlock avoidance**).

Application programmers need to be aware that, depending on the threads implementation and the underlying hardware, concurrency may be more apparent than real for many applications. If threads are being timesliced on a *single* processor, non-blocking activities will not go any faster because they are multi-threaded. In fact, given the extra overhead of a threads implementation, they might even be slower. On the other hand, if multiple threads are carrying out activities that may block, then multi-threading will probably be beneficial.

In summary, the developer must consider the following questions in order to decide whether an application will benefit from multi-threading:

- Are the threaded operations likely to block, for example, because they make blocking I/O calls? If so, then multi-threading is likely to be beneficial in any implementation or hardware environment.
- Can the underlying hardware and implementation support threads on more than one processor within a single process? If not, then multi-threading cannot achieve *real* parallelism for *processor intensive* operations.
- Even if the answer to both of the first two questions is yes, will the use of a timeslicing thread scheduling policy enable more equitable distribution of resources among contending clients? If so, then multi-threading may be beneficial.

If programming is to be thread-safe, the code must be structured such that it can be executed simultaneously by multiple threads. For thread-safe programming, you have to use and implement thread-safe functions. To make a function thread-safe you have to do the following:

Use mutexes and condition variables
: These work on one appointed thread and are prepared by the application programmer. They are used to synchronize threads on the application level. They are described in the section **Threads synchronization objects**.

Make global data thread specific
: This allows a thread to allocate and maintain private static data that needs no protection by mutexes against concurrent access by other threads. This is described in the section **Thread specific data**.

Use thread-safe libraries
: The functions in thread-safe libraries are either thread-reentrant as such or they are "jacketed". For some functions the library may provide a thread-safe renamed "_r" function. The programmer can use this function in place of the non-reentrant function. For example you can use `strtok_r(3)` which is reentrant instead of `strtok(3C)` which is not reentrant. The thread-safe system libraries are described in the section **Thread-safe system libraries**.

Multi-threaded programming imposes restrictions and specific semantic issues for certain system interfaces [for example, signal handling, `fork(2)` and `exec(2)`] and system libraries. Using thread-safe (system) libraries within a non-threaded program is undefined. Using thread-unsafe (system) libraries within a multi-threaded program is possible if controlled appropriately by the application.

Any executable program (or object or library) built with the compiler's `thread` option (or linked with modules of the threaded C runtime support) becomes a multi-threaded program (or object or library) with all side effects on functionality even if it does not call a threads API. On the other hand, a multi-threaded program with exactly one thread *is not equivalent* to a non-threaded program.

DCE Threads

The Distributed Computing Environment (DCE) includes DCE Threads [**DCE Threads**], a user-space implementation of functions to support multiple flows of control, called *threads*, within a UNIX process. DCE Threads define system interfaces to support the source portability of threaded applications.

DCE Threads are built on draft 4 of the IEEE project POSIX 1003.4a [**POSIX.4a/D4**]. Draft 4 is no longer available, the subsequent draft 10 evolved into IEEE Standard 1003.1c [**POSIX.1**], known as *Pthreads*.

The specific functional areas covered by DCE Threads and their scope includes:

Thread management
: Including the creation, control and termination of multiple flows of control (threads) in the same process under the assumption of a common, shared address space.

Synchronization primitives
: Optimized for tightly coupled operation of multiple flows of control in a common, shared address space.

Harmonization with the existing system interfaces
: The resolution of the impact of multiple flows of control (threads) onto the system interfaces previously defined under the assumption of one flow of control. This is the issue of thread-safe system libraries (see the section **Thread-safe system libraries**).

Exception handling
: A program using DCE Threads can evaluate the result of DCE Threads API calls via the status values returned or via the exception-returning interface. Exceptions enable routines to ignore status returns when other parts of the program are handling errors.

There are no administrative tasks involved with DCE Threads except software installation.

DCE Threads are available as the `SIthreads` 5.44B package for Reliant UNIX 5.44B and `SIthread` 5.45A for Reliant UNIX 5.45A. It supports the DCE threads API and a set of associated thread-safe libraries.

The implementation of DCE Threads resides entirely in user land without interaction with the Reliant UNIX kernel. DCE Threads include logic for timesliced, priority-driven, preemptive threads scheduling, and for avoiding threads blocking due to blocking system calls.

The following sections summarize DCE Threads. The notation is borrowed from [**POSIX.4a/D10**].

Include files

All source using DCE Threads must include the header file

```
#include <pthread.h>;
```

DCE Threads also provide an exception-returning interface as an extension to the basic threads functionality. This requires the include file

```
#include <pthread_exc.h>;
```

instead of `pthread.h`.

Symbolic constants controlling DCE Threads features

The following symbolic constants are defined as unspecified values in `pthread.h`:

Symbolic constants	Defined	Description
`_POSIX_THREADS`	yes	The implementation supports the threads option.
`_POSIX_THREADS_REALTIME_SCHEDULING`	yes	The implementation supports the attributes and functions that all priority scheduling.
`_POSIX_THREADS_PER_PROCESS_SIGNALS_1`	yes	The implementation supports per-process delivery of asynchronc Asynchronous delivery of signals on a per-thread basis is not sup
`_POSIX_THREAD_ATTR_STACKSIZE`	yes	The implementation supports the thread stack size attribute optic

Compiler issues

The Reliant UNIX C/C++ compiler CDS++ since 1.0 [**C/C++**] supports compiling and linking of applications using DCE Threads as specified below.

Note: DCE Threads and the thread-safe libraries included are incompatible with `c89` and strict ANSI mode in C and C++.

Pre-processor definitions and search paths

The compiler's `-K thread` option indicates usage of DCE Threads with the following defaults:

Predefined names

The preprocessor name `__SNI_SVR4`, `__SNI_THREAD_SUPPORT`, `__SNI_DCOSX`, and `_REENTRANT` are always defined.

Include paths

The compiler automatically adds the path `/opt/thread/include` to the list of standard directories searched for include files (before `/usr/include` and `/usr/include/sys`).

Standard libraries

The compiler automatically adds the library option `-l cma` for the DCE Threads API library `libcma.so` to the link process (unless suppressed by another option) and search paths for thread-safe system libraries like `/opt/thread/lib[64s]`, `/opt/CDS++/lib[64s]/thread`, `/usr/ccs/lib[64s]/thread`, `/usr/lib[64s]/thread`.

Note:

Check your use of `LD_RUN_PATH` and `LD_LIBRARY_PATH` in the compilation and runtime environment to avoid unintended interference between thread-safe and thread-unsafe libraries!

The compiler also adds the reference `/opt/thread/lib[64s]/libc_r.so.1` for the shared standard C library. This (absolute) reference is used at runtime by the runtime loader.

Note:

Building a library with `-K thread` makes that library unusable in a non-threaded environment!

Startup routines

If `-K thread` is specified, the compiler automatically links startup routines to the executable that performs proper initialization of DCE threads and thread-safe libraries prior to calling `main`.

Module *)	If ...
`/opt/CDS++/lib[64s]/pthr_dce_c.o`	`cc` command without `-p` option
`/opt/CDS++/lib[64s]/pthr_dce_mc.o`	`cc` command with `-p` option
`/opt/CDS++/lib[64s]/pthr_dce.o`	`CC` command without `-p` option
`/opt/CDS++/lib[64s]/pthr_dce_m.o`	`CC` command with `-p` option

*) These modules and their location do not represent an external user interface for the compiler.

DCE Threads provide the following functions for initialization in the modules mentioned above. They need to be called in the order given below and *before* any other DCE Threads routine is used. Use of these functions outside of startup is undefined.

```
void cma_init(void);
```

Initializes the DCE Threads library `libcma.so`.

```
void set_pthread_symb(void);
```

Sets up various callbacks from `libc_r.so` back into `libcma.so`. These are needed to make the standard C library thread-safe.

```
void libc_r_init(void);
```

Initializes the thread-safe `libc_r.so` library.

Profiling

Profiling requires relinking of the threaded application with the `-p` compiler option. The startup code starts a separate thread that collects the profile data into a text file named `mon.`*program* in the current working directory. Function call counts are not available. The created file `mon.out` is of no use.

```
% cc -p -K threads -o program files ...
```

Debugging

The symbolic `dbx` debugger that comes with the Reliant UNIX C compiler CDS++, is threads-aware and supports various actions to display and control threads state and execution. See [**C/C++**] for details.

DCE Threads offer the following functions in `libcma_ext.so` to `dbx` to display the thread state. The declaration and explanation of most of the arguments is contained in `cma_deb_ext.h` (this file is not part of the DCE Threads package). Use of these functions outside of `dbx` is undefined.

```
#include <cma_deb_ext.h>;
void cma_init_debug(deb_function_vector_t *deb_fv_p,
     cma_function_vector_t **a_cma_fv_p);
```

Sets up the use of DCE Threads debugging information via the mutual function pointers in the arguments. The debugger will pass a pointer to its function vector *deb_fv_p*, through which DCE Threads will access the debugger specific routines. DCE Threads will return the pointer *a_cma_fv_p* to its function vector that the external debugger will use. Those functions are:

```
#include <dce/cma.h>;
void cma_debug_cmd(char *cmd, ...);
```

Parse commands in *cmd* and dispatches to execution code.

```
#include <cma_deb_ext.h>;
int cma__deb_getcontext(unsigned int thread_num,
    cma_t_deb_thread_context *context_p);
```

Gets the context of the thread with the given sequence number *thread_num* into the area pointed to by *context_p* in order to switch to another thread. This routine cannot be called for the current thread (i.e. the thread which took a breakpoint).

```
unsigned int cma__deb_get_self_sequence(void);
```

Returns the sequence number of the currently running thread.

```
int cma__deb_is_thread_valid(unsigned int thread_num);
```

Verifies whether the given thread sequence number *thread_num* refers to a known thread no matter what state that thread is in.

In addition, `dbx` might use the following symbols to check their existence or retrieve their address (but not the data):

```
cma__g_def_tcb               /* TCB of main thread */
cma__g_known_threads         /* List of all threads */
cma__transfer_thread_ctx     /* To dispatch threads */
cma__transfer_thread_ctx_pc
pthread_mutexattr_default
```

Name space for DCE Threads

DCE Threads objects (data types) and function calls, with few exceptions, follow the naming convention

`pthread`[*_object*]`_t`, `pthread`[*_object*]*_operation*[`_np`]

Here, *object* optionally specifies the object (data type) more closely and *operation* specifies the operation on *object*. The suffix `_np` indicates a non-portable extension with respect to [**POSIX.4a/D4**].

The DCE Threads implementation defines (and reserves) all identifiers with prefix `pthread`, `ptdexc`, `PTHREAD`, `cma`, and `CMA`.

Implementation limits

Description	Value
Threads within a process	Limited by available memory in process
Mutexes within a process	Limited by available memory in process
Condition variables within a process	Limited by available memory in process
Keys for thread-specific data (TSD) in a process	Limited by available memory in process
Minimum threads stack size	32K (plus 16K internally reserved space
Minimum threads guard size	16K

The DCE Threads implementation allocates 2K per thread (in advance and reused later on) for a thread control block. Each TSD key occupies the memory space needed for a pointer per thread actually accessing that key. Each mutex occupies 72 bytes, each condition variable occupies 40 bytes of memory.

The DCE Threads implementation allocates at least the minimum stack and guard size per thread from the heap. Note that a thread might *increase* both sizes via the associated attribute. The implementation allocates memory for threads stacks in advance and reuses the space freed by terminated threads. Because of this dynamic nature, a precise formula of memory consumption for threads stacks is not available. A conservative estimate is the average stack size per thread (application dependent, at least 64K) times the peak number of threads during the lifetime of the process plus an amount of 25% for memory fragmentation.

DCE Threads data types

DCE Threads specify the following objects (data types) in `pthread.h`:

Data types	Internal layout *)	Description

`pthread_t`	□
	□
	`struct { □`
	`void *field1; □`
	`short field2; □`
	`short field3; □`
	`};`
		Thread ID

`pthread_attr_t`	Like `pthread_t`	Threads attribute object
`pthread_cond_t`	Like `pthread_t`	Condition variable
`pthread_condattr_t`	Like `pthread_t`	Condition variable attribute object
`pthread_key_t`	`int`	Thread specific data (TSD) key

`pthread_once_t`	□
	□
	`struct { □`
	`int field1; □`
	`void *field2; □`
	`short field3; □`
	`};`
		Once-only execution

`pthread_mutex_t`	Like `pthread_t`	Mutual exclusion lock
`pthread_mutexattr_t`	Like `pthread_t`	Mutual exclusion lock attribute object
`pthread_addr_t`	`void *`	Address for arguments of called functions
`pthread_destructor_t`	`void *`	Address of the destructor function for TSD keys

pthread_initroutine_t	void *	Address of init routine for once-only execution
pthread_startroutine_t	void *	Address of thread start routine

*) The data types are opaque for the programmer using DCE Threads. With the exception of pthread_t and the address types, there are no defined comparison or assignment operators for these data types.

DCE Threads API

In the following sections, the notation of function arguments of the form

type name = *value*

indicates that *value* may safely be used instead of *name*.

Thread attributes

DCE Threads attributes are collected into a thread attributes object defined by data type pthread_attr_t. A thread attributes object is created and destroyed by the functions

```
int pthread_attr_create(pthread_attr_t *attr);
int pthread_attr_delete(pthread_attr_t *attr);
```

All thread attributes in a thread attributes object are set by a function of the form

```
int pthread_attr_setName(pthread_attr_t *attr, Type t);
```

and retrieved by a function of the form

```
Type pthread_attr_getName(pthread_attr_t attr);
```

where *Name* and *Type* are from the table below.

Type and name	Value(s) with default
int inheritsched	PTHREAD_INHERIT_SCHED (default), PTHREAD_DEFAULT_SCHED
int prio	PRI_OTHER_MIN <= prio <= PRI_OTHER_MAX (SCHED_OTHER) □ PRI_FIFO_MIN <= prio <= PRI_FIFO_MAX (SCHED_FIFO) □ PRI_RR_MIN <= prio <= PRI_RR_MAX (SCHED_RR) □ PRI_FG_MIN_NP <= prio <= PRI_FG_MAX_NP (SCHED_FG_NP) □ PRI_BG_MIN_NP <= prio <= PRI_BG_MAX_NP (SCHED_BG_NP) □ Default is the midpoint between minimum and maximum.
int sched	SCHED_FIFO, SCHED_RR, SCHED_OTHER, SCHED_FG_NP, SCHED_BG_NP
long stacksize	>= 0
long guardsize	>= 0

Note: Access functions for guardsize have _np extension.

Thread creation, termination, and scheduling

This includes the creation, control and termination of multiple flows of control (threads) in the same process under the assumption of a common, shared address space.

```
int pthread_create(pthread_t *thread,
    pthread_attr_t attr = pthread_attr_default,
    pthread_startroutine_t routine,
    pthread_addr_t arg);
```

Creates a new thread with attributes *attr*, executing *routine* with *arg*.

```
int pthread_detach(pthread_t *thread);
```

Indicates to reclaim all resources (including termination status) when *thread* terminates.

```
boolean32 pthread_equal(pthread_t *t1, pthread_t *t2);
```

Compares thread IDs *t1* and *t2* for equality (implemented as macro).

`void pthread_exit(pthread_addr_t status);`

Terminates the calling thread (does not return). *status* will be returned to a thread synchronizing with the terminating thread.

`int pthread_join(pthread_t thread, pthread_addr_t *status);`

Synchronizes with (waits for) the termination of the specified *thread* getting its termination *status*.

`pthread_t pthread_self(void);`

Returns the thread ID of the calling thread.

`void pthread_yield(void);`

Releases control in the calling thread to another thread.

`int pthread_getprio(pthread_t thread);`

Gets the scheduling priority currently set for *thread*.

`int pthread_setprio(pthread_t thread, int priority);`

Sets the new scheduling *priority* for *thread*.

`int pthread_getscheduler(pthread_t thread);`

Gets the scheduling policy currently set for *thread*.

```
int pthread_setscheduler(pthread_t thread, int policy,
    int priority);
```

Sets the new scheduling *policy* and *priority* for the calling thread.

```
void atfork(void (*arg)(void), void (*prepare)(void),
     void (*parent)(void), void (*child)(void));
```

Registers functions to be called during `fork(2)` execution. *prepare* functions are called before `fork(2)` in reverse order of registration (LIFO), *parent* and *child* functions are called in the parent and child respectively in order of registration (FIFO). *arg* is passed as the only argument to all these routines.

Thread execution scheduling

DCE Threads implement logic for timesliced, priority-driven, preemptive threads scheduling. DCE Threads are scheduled according to their scheduling *priority* and how the scheduling *policy* treats those priorities.

Timeslicing is the mechanism that ensures that every thread is allowed time to execute by preempting running threads after 2 ticks. DCE Threads reserve the interval timer `ITIMER_VIRTUAL` (and thus the signal `SIGVTALRM`) for timeslicing threads.

DCE Threads reads the timeslice length during execution of the first `pthread_create()` from the environment variable

`CMA_TIMESLICE=`*n* # 40 < n <= 200 [msec], increment 20

The length is per default 200 msec. Values outside of this interval are set to the corresponding interval boundary.

The *inherit scheduling* attribute specifies whether a newly created thread inherits the scheduling attributes (priority and policy) of the creating thread (the default), or uses the scheduling attributes stored in the thread attributes object set by `pthread_attr_setinheritsched()`.

The DCE Threads scheduling policies guarantee that even low priority threads will get to run. This will provide at least the benefit of fair access to processing time for multiple threads, even when no *real* parallelism is provided for multiple threads of execution.

The *scheduling policy* attribute describes the overall scheduling policy of the threads in the application. A thread has one of the following scheduling policies:

`SCHED_FIFO` (First In, First Out)
: The highest-priority thread runs until it blocks. If there is more than one thread with the same priority, and that priority is the highest among other threads, the first thread to begin running continues until it blocks. Priority does not change within this policy unless changed by `pthread_setprio()`.

`SCHED_RR` (Round Robin)
: The highest-priority thread runs until it blocks; however, threads of equal priority, if that priority is the highest among other threads, are timesliced. A preempted thread goes to the tail of the ready queue for its priority. Priority does not change within this policy unless changed by `pthread_setprio()`.

`SCHED_FG_NP` and `SCHED_BG_NP`
: All threads are timesliced. All threads, regardless of priority, receive some scheduling. Therefore, no thread is completely denied execution time. `SCHED_FG_NP` has a higher priority band than `SCHED_BG_NP`. However, both can be denied execution time by `SCHED_FIFO` or `SCHED_RR` threads. Priority does not change over time within these policies unless changed by `pthread_setprio()`.

`SCHED_OTHER` (Default)
: Is identical to `SCHED_FG_NP`.

`pthread_attr_setsched()` sets the scheduling policy attribute in the thread attributes object, which establishes the scheduling policy of a new thread when it is created. `pthread_setscheduler()` changes the scheduling policy (and, at the same time, the scheduling priority) of an existing thread.

The *scheduling priority* attribute specifies the execution priority of a thread. This attribute is expressed relative to other threads on a continuum of minimum to maximum for each scheduling policy. A thread's priority falls within a priority range, which is implementation defined.

`pthread_attr_setprio()` sets the scheduling priority attribute in the thread attributes object, which establishes the execution priority of a new thread when it is created. `pthread_setprio()` changes the scheduling priority attribute of an existing thread.

Thread cancellation

The thread cancellation mechanism allows a thread to terminate the execution of any other thread in the process in a controlled manner. Each thread maintains its own cancelability state. The target thread is allowed to hold cancellation requests pending and to perform application-specific cleanup processing when it acts upon the notice of cancellation.

`int pthread_cancel(pthread_t` *thread*`);`

Places a cancellation request in the specified *thread*. *Note*: Canceled threads terminate with a status of –1.

`void pthread_testcancel(void);`

Requests delivery of a pending cancellation request (if any) in the calling thread. *Note*: This function *is* a cancellation point.

`int pthread_setcancel(int` *state*`);`

Sets the general cancellation *state* of the calling thread to one of `CANCEL_ON` (the default for each newly created thread) or `CANCEL_OFF` and returns the previous state.

`int pthread_setasnyccancel(int` *state*`);`

Sets the asynchronous cancellation *state* of the calling thread to one of `CANCEL_ON` or `CANCEL_OFF` (the default for each newly created thread) and returns the previous state.

```
void pthread_cleanup_push(void routine,
     pthread_addr_t arg);
```

Pushes *routine* and *arg* onto the calling thread's cancellation stack. *Note*: Push and pop operations must appear at the same lexical level (same level of { and } in C language).

`void pthread_cleanup_pop(int` *execute*`);`

Pops the top item from the calling thread's cancellation stack and executes it, if *execute* is nonzero.

Cancellation points

Cancellation points are points inside a function where a thread must act on any pending cancellation request when cancelability is enabled. Every function that calls a function defined to contain cancellation points itself becomes a cancellation point.

DCE Threads define cancellation points to occur when a thread is executing the following functions. In

addition, cancellation points exist within the handling of the timeslice interrupt (see the section **Thread execution scheduling**) and the DCE I/O wrappers for blocking calls (see the section **Blocking functions**).

Functions defined as cancellation points	
DCE Threads API calls	
pthread_cond_timedwait()	pthread_cond_wait()
pthread_delay_np()	pthread_join()
pthread_testcancel()	pthread_setasynccancel()
sigwait()	

Note: pthread_mutex_lock() is not a cancellation point.

Async-cancel safety

A target thread may disable cancellation altogether or set it to *asynchronous cancelability*. During a period when asynchronous cancellation is enabled, the target thread must only execute code that does not modify global state and may only call routines that are explicitly stated as *async-cancel safe* in their documentation.

A function is said to be *async-cancel safe* if it is written in a way that entering it from a thread with asynchronous cancelability enabled will not cause any global state to be violated. The DCE Threads API defines only pthread_cancel() to be async-cancel safe.

Threads synchronization objects

Mutex attributes

Attributes of mutexes are collected into a mutex attributes object defined by data type pthread_mutexattr_t. A mutex attributes object is initialized and destroyed by the functions

```
int pthread_mutexattr_create(pthread_mutexattr_t *attr);
int pthread_mutexattr_delete(pthread_mutexattr_t *attr);
```

All mutex attributes are set in a mutex attributes object by a function of the form

```
int pthread_mutexattr_setName_np(pthread_mutexattr_t *attr, Type t);
```

and retrieved by a function of the form

```
Type pthread_mutexattr_getName_np(pthread_mutexattr_t attr);
```

where *Name* and *Type* are from the table below.

Type and name	Value(s) with default
int kind	MUTEX_FAST_NP (default), MUTEX_RECURSIVE_NP, MUTEX_NONRECURSIVE_NP

Mutex management

Mutexes are synchronization variables that allow locking of critical code sections against simultaneous execution by more than one thread to avoid data races. Only one thread can lock (acquire) a mutex and proceed. Other threads will wait synchronously until a locked mutex is unlocked (released) by the thread that owns it.

```
int pthread_mutex_init(pthread_mutex_t *mutex,
    pthread_mutexattr_t attr = pthread_mutexattr_default);
```

Initializes the indicated *mutex* with attributes *attr* or default attributes.

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Destroys the indicated *mutex*.

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

Acquires the indicated *mutex* (waiting for it if necessary).

```
int pthread_mutex_trylock(pthread_mutex_t *mutex);
```

Tries to acquire the indicated *mutex* (never waiting for it).

```
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Releases the (previously acquired) indicated *mutex*.

Condition variable attributes

Attributes of condition variables are collected into a condition variable attributes object defined by data type `pthread_condattr_t`. A condition variable attributes object is initialized and destroyed by the functions

```
int pthread_condattr_create(pthread_condattr_t *attr);
int pthread_condattr_delete(pthread_condattr_t *attr);
```

Condition variable attributes objects do not expose modifiable attribute values.

Condition variable management

Condition variables, in conjunction with mutexes, allow threads to wait for some arbitrary condition to occur, i.e. to be signaled on this condition variable. Condition variables are stateless in the sense that signals on them are not remembered if no thread is waiting.

```
int pthread_cond_init(pthread_cond_t *cond,
    pthread_condattr_t attr = pthread_condattr_default);
```

Initializes the indicated condition variable *cond* with attributes *attr* or default attributes.

```
int pthread_cond_destroy(pthread_cond_t *cond);
```

Destroys the indicated condition variable *cond*.

```
int pthread_cond_signal(pthread_cond_t *cond);
```

Unblocks at least one thread currently blocked in the condition variable *cond*.

```
int pthread_cond_broadcast(pthread_cond_t *cond);
```

Unblocks all threads currently blocked in the condition variable *cond*.

```
int pthread_cond_wait(pthread_cond_t *cond,
    pthread_mutex_t *mutex);
```

Blocks on the condition variable *cond* protected by *mutex*. *Note*: This function is a cancellation point.

```
int pthread_cond_timedwait(pthread_cond_t *cond,
    pthread_mutex_t *mutex, struct timespec *abstime);
```

Blocks on the condition variable *cond* protected by *mutex* no longer than the specified *abstime*. *Note*: This function is a cancellation point.

Thread specific data (a.k.a. per-thread context)

The thread-specific data (TSD) interface can be viewed conceptually as a two-dimensional array with *keys* serving as the row index and thread IDs as the column index. Thus the keys are shared among all the threads while each thread has an individual data *value* (usually a pointer) associated with each key.

```
int pthread_keycreate(pthread_key_t *key,
    pthread_destructor_t destructor);
```

Creates and returns a new data *key* and associates the *destructor* function with that key for the calling thread. The *destructor* function is called with the data value associated with *key* as argument upon thread exit.

```
int pthread_getspecific(pthread_key_t key,
    pthread_addr_t *value);
```

Returns *value* associated with *key* in the calling thread (or `NULL`).

```
int pthread_setspecific(pthread_key_t key,
    pthread_addr_t value);
```

Sets *value* associated with *key* in the calling thread.

Once-only execution

```
int pthread_once(pthread_once_t *once_control,
    pthread_initroutine_t routine);
```

Executes *init_routine* exactly once.

Non-portable extensions

The DCE Threads API specifies the following set of non-portable functions. Non-portable means they do not exist in [**POSIX.4a/D4**].

```
int pthread_get_expiration_np(struct timespec *delta,
    struct timespec *abstime);
```

Returns the sum of *delta* and the current absolute time in *abstime*.

```
int pthread_delay_np(struct timespec *interval);
```

Delays execution of the calling thread for the specified *interval*.

```
int pthread_lock_global_np(void);
```

Acquires the global lock (waiting for it if necessary).

```
int pthread_unlock_global_np(void);
```

Releases the global lock.

```
void pthread_signal_to_cancel_np(sigset_t *sigset,
     pthread_t *thread);
```

Requests that the specified *thread* be canceled if one of the signals from *sigset* is received by the process. *Note*: *thread* is saved globally and overwritten by each call.

Extensions for the test environment

The following functions provide functionality for the threads test environment. Use of this functions by applications is undefined.

```
int pthread_delay_unsigned_np(unsigned sec);
```

Delays execution of the calling thread for the specified interval *sec*. Compares to `pthread_delay_np()` with another type of parameter.

Threads execution timing

The following functions provide functionality to measure the timing of threads execution in the areas of scheduling and I/O.

DCE Threads collect timing information about threads scheduling if the environment variable

`CMA_TRACESCHEDTIMES=`*n*

is set during execution of the first `pthread_create()`. The value *n* (at least 64) defines the number of `pthread_schedtime` entries in an internal circular list. Each entry holds an identification of the thread and a timestamp of its scheduling.

```
int pthread_getschedtimes_np(pthread_schedtime *buffer);
```

reads the contents of that circular list into *buffer* and returns the number of entries. It always returns at most the last *n* entries collected independently from the last call to retrieve the buffer. The buffer is allocated by the caller and must have size *n* * `sizeof(pthread_schedtime)`. The entries are always ordered chronologically.

DCE Threads collect timing information about threads execution in I/O calls if the environment variable

`CMA_TRACEIOTIMES=`*n*

is set during execution of the first `pthread_create()`. The value *n* (at least 64) defines the number of entries `pthread_iotime` in an internal circular list. Each entry holds an identification of the thread, a timestamp of the IO call, the interval spent in the IO in msec (max. 65.000), the number, file descriptor return value and errno value of the IO call. The following I/O calls are supported: `read(2)`, `readv(2)`, `write(2)`, `writev(2)`, `getmsg(2)`, `putmsg(2)`, `ioctl(2)`.

`int pthread_getiotimes_np(pthread_iotime *`*buffer*`);`

reads the contents of that circular list into *buffer* and returns the number of entries. It always returns at most the last *n* entries collected independently from the last call to retrieve the buffer. The buffer is allocated by the caller and must have size *n* * `sizeof(pthread_iotime)`. The entries are not necessarily chronologically ordered.

Thread-safeness and reentrance

A function may be "reentered" by several mechanisms: by calls from multiple threads, by a call from an asynchronous signal handler that interrupted the function, by direct or indirect recursion. Thread-safe indicates reentrance with respect to simultaneous calls by multiple threads. A function that is reentrant with respect to asynchronous signals is *async-signal* safe.

Data races

A program has a data race if it is possible for a thread to modify an addressable location at the same time that other threads are accessing the same location. Reading the same location in memory concurrently does not represent a data race.

Programs and thread-safe functions must avoid data races by using the synchronization primitives of DCE Threads. However, thread-safe functions do not protect access to memory under control of the caller. For example, the calling program must avoid the data race produced by two simultaneous calls to `memcpy(3C)` with overlapping destination areas.

Threads and process resources

Each thread has only a small individual state: the register set, program counter, stack pointer and signal mask, the `errno` variable and the thread-specific data (if any). All other resources of a process are shared by all the threads.

Address space
: Static data is accessible to all threads and requires appropriate protection against data races.

File descriptors
: Concurrent access to the same file by multiple threads (for example, reading the file record by record) must control the file offset before each atomic access.

Timers
: Timers are process-global. The corresponding expiration signal is sent to the process, not a particular thread. Thus, threads need to cooperate to set per-thread timers.

Signal handlers
: Are discussed below in a separate section.

Environment
: Arguments (`argv`) and environment (`environ`) are part of the address space and need protection against data races. *Note*: The `getenv(3C)` function is thread-unsafe but does not have a `getenv_r` variant.

User ID and working directory
: Changing the user ID or working directory in one thread might lead to denial of access to parts of the file system in other threads.

The errno variable

Usage of a thread-aware `<errno.h>` in any multi-threaded program is required to coordinate the access of the threads to the global `errno` value. It transforms all references to `errno` into `*cma_errno()`. `cma_errno()` returns `&errno`. Thus, the currently running thread in the process actually uses `errno`. Thread scheduling saves the value of `errno` into the thread control block of the descheduled thread and

loads errno from the thread control block of the newly scheduled thread. This also allows thread-unaware libraries or modules to use `errno` directly.

Note that the DCE Threads implementation supports at most one running thread per process at a time.

Serializability

Most simultaneous calls to thread-safe functions appear to be "atomic", i.e. as if executed in a (arbitrary) serial order. However, this need not be the case. For example, a call to a function *f* that adds two items *A* and *B* to a list, and another simultaneous call that adds *C* and *D* may produce a list in the order *A*, *C*, *B*, *D*. This is not possible with serial execution of the two calls or if the add operation is defined to be atomic. Thread-safe functions should specify any behaviors that are not serializable.

The standard I/O functions like `printf(3S)` are individually thread-safe and simultaneous calls using the same `FILE` object are serialized. However, the calling application or library has to make a series of such atomic calls and protect the `FILE` object against a simultaneous `write(2)` to the underlying file.

Functions reading *successive* entries in a file are usually not thread-safe since information about which entry was last read is not maintained on a per-thread basis. Consequently, the application program must synchronize the use of these functions.

Synchronization strategies and deadlock avoidance

Threads' synchronization strategies must avoid data races. They vary in the degree to which they allow threads to overlap operation.

Structured code and data locking

Structured *code* locking assumes that shared data is only accessed by a set of specific functions. Typically, there is a single mutex ("monitor lock") associated with this set of functions that is acquired upon entering any of them and released on return. Thus only one thread at a time can execute any of the functions.

Structured *data* locking assumes that shared data is only accessed by a set of specific functions *and* can be grouped into disjoint objects sharing no state. It associates a mutex ("object lock") directly with each object that is acquired whenever the object is manipulated. Operations on different objects can proceed concurrently.

A monitor or object lock needs to be released when nesting of these locks (needing another while already holding one), reentrant operations (reacquiring a lock already held), long operations (I/O to read in object data) or calls to functions of unknown locking behavior (outside of the current context) occur. This is necessary to enable concurrency and to avoid deadlocks. The shared data must be consistent when the lock is released and reevaluated before proceeding when the lock is reacquired.

Deadlock, lock ordering, resource deadlocks

There are various dangers of deadlock associated with synchronization issues. *Self*-deadlock or *recursive* deadlock occurs when a thread tries to acquire a lock that it already owns. *Nested* deadlock occurs where a thread holding the monitor lock of function set *A* blocks on function set *B*, but the path to release *B* goes through *A*. All of these can be avoided by releasing a monitor or object lock before calling external functions.

Lock-ordering deadlock occurs where a thread holds *A*'s lock and tries to acquire *B*'s lock while another thread holds *B*'s lock and tries to acquire *A*'s lock. This can be avoided by defining a lock order or lock hierarchy in which locks are acquired.

Resource deadlock occurs where a thread needs a second resource of the same type to proceed but all such resources are already allocated as first resources by other threads and none is willing to give one up. The solution is to first trying to acquire all resources needed to progress and if that is not possible give them all up and restart.

Thread-safe system libraries

DCE Threads include thread-safe variants of most (but not all) system libraries. They are named as `lib`*name*`_r.so` and `lib`*name*`_r.a`

Thread-safe libraries exist in addition to their thread-unsafe companions in directories known to the Reliant UNIX C compiler (see the section **Compiler issues**). The definitive list of thread-safe libraries and thread-safe

functions is part of the release notes for the DCE Threads package. In the presence of the `-K thread` option, the compiler maps the usual `-l`*name* indication for libraries appropriately.

Thread-safe system libraries included in DCE Threads		
libc.so	libc.so.1	libgen.a
libm.a	libmalloc.a	libnsl.so
libnsl_d.so	libnsl_i.so	libresolv.so
libsocket.so	resolv.so	tcpip.so
libabi.so		

All functions in the system library `libdl.so` can also be used safely in a multi-threaded application.

For the system functions which are, in principle, thread-unsafe, DCE Threads offer thread-safe variants:

Thread-Safe versions of thread-unsafe functions		
asctime_r()	ctime_r()	endhostent_r()
getgrgid_r()	gethostent_r()	getpwnam_r()
getpwuid_r()	localtime_r()	rand_r()
readdir_r()	sethostent_r()	strtok_r()

Using thread-unsafe libraries

A multi-threaded application might use thread-unsafe functions or libraries if it controls the calls in a way that avoids the issues as explained in the section **Thread-safeness and reentrance**. This can be achieved by appropriate locking prior to such calls or by always calling from one and the same (main) thread.

Signal handling

Applications of DCE Threads must handle signals differently from standard non-threaded Reliant UNIX applications.

A signal is considered *generated* for a process or thread as a result of some event. It is considered *delivered* when the assigned signal action for the process or thread is taken. In the interval between generation and delivery, the signal is *pending*. A signal can be *blocked* (*masked*) from delivery by a process or thread via a signal mask for that process or thread.

Synchronous signals are generated by internal events associated with program execution (e.g. illegal memory reference) and delivered to the affected thread. *Asynchronous* signals are generated by external events (like user interaction or timers) and delivered once to one (waiting) thread in the process that has that signal unmasked.

DCE Threads implement a model for signal delivery where asynchronous signals are pending on the process and delivered to the first thread that unblocks for a particular pending signal, whereas synchronous signals are delivered to the thread generating them. Thus, asynchronous signals are delivered on a per-process basis and synchronous signals are delivered on a per-thread basis.

For synchronous signals, DCE Threads support per-thread signal handlers via `sigaction()` and per-process signal masks via `sigprocmask()`. For asynchronous signals, DCE Threads support a per-process signal mask via `sigprocmask()` and handling of such signals in a dedicated thread via `sigwait()`.

```
#include <pthread.h>;
int sigwait(sigset_t *sigset);
```

Waits for an *asynchronous* signal in *sigset* and returns the signal delivered. *sigset* must not contain one of the signals `SIGKILL`, `SIGSTOP`, `SIGILL`, `SIGTRAP`, `SIGIOT`, `SIGEMT`, `SIGFPE`, `SIGBUS`, `SIGSEGV`, `SIGSYS`, `SIGPIPE`, `SIGXCPU`, and `SIGXFSZ`. Note that `SIGVTALRM` is reserved.

```
#include <signal.h>;
```

```
int sigprocmask(int how, const sigset_t *set = NULL,
    sigset_t *oset = NULL);
```

Blocks/unblocks a *synchronous* signal in *set* for the calling thread or an *asynchronous* signal for the process and returns the previous signal mask in *oset*.

```
#include <signal.h>;
int sigaction(int sig, const struct sigaction *act,
    struct sigaction *oact);
```

Defines signal handling action *act* for *synchronous* signal *sig* and returns the previous signal handling in *oact*.

Note: The use of the signal management functions signal(2), sigset(2), sighold(2), sigrelse(2), sigignore(2), sigpause(2) is unspecified in multi-threaded processes.

Signal SIGVTALRM

DCE Threads reserve the interval timer ITIMER_VIRTUAL for timeslicing threads. Therefore, DCE Threads install a handler for SIGVTALRM which thus becomes unavailable for use by the application.

Async-signal safety

A function is called *async-signal safe* if it may be called, without restrictions, from signal-catching functions invoked asynchronously with threads execution. All functions are considered to be unsafe with respect to signals unless explicitly declared as async-signal safe. In the presence of signals, all functions will behave as defined when called from or interrupted by a signal-catching function, with a single exception: when a signal interrupts an unsafe function and the signal-catching function calls an unsafe function, the behavior is undefined.

Note: DCE Threads do not declare any function of the DCE Threads API or the thread-safe system libraries to be async-signal safe. DCE Threads API functions and system libraries (thread-safe or not) cannot be called safely in a signal handler.

In particular, you cannot use longjmp(3C) safely in signal handlers. □

Blocking functions

DCE Threads implement mechanisms to make blocking calls thread-synchronous. This ensures that only the calling thread is blocked and the process remains available to execute other threads. For blocking I/O calls involving file descriptors, DCE Threads provide I/O wrappers to control access to all file descriptors.

Thread-blocking functions		
accept(3N)	bind(3N)	close(2)
connect(3N)	getmsg(2)	getpmsg(2)
listen(3N)	open(2)	poll(2)
putmsg(2)	putpmsg(2)	read(2)
readv(2)	recv(3N)	recvfrom(3N)
recvmsg(3N)	select(3C)	send(3N)
sendmsg(3N)	sendto(3N)	write(2)
writev(2)		

If a request on a file descriptor would block, the DCE Threads implementation suspends the calling thread and reschedules it after the blocking condition on the file descriptor has gone away. As a result, those calls never return the errno value EINTR.

However, blocking access to character devices not supporting poll(2) will block the process.

The following blocking calls that do not involve file descriptors will block the process, not only the calling thread.

Process-blocking functions

msgsnd(2)	msgrcv(2)	pause(2)
semop(2)	sigpause(2)	sigsuspend(2)
system(3S)	wait(2)	waitid(2)
waitpid(2)		

Fork/Exec/Exit

DCE Threads only propagate the thread calling fork(2) to the new process. Other threads are eliminated silently and without cleaning up any resources locked by them either in application or in runtime library code. This cleanup occurs during exec(2) when the address space is reinitialized.

Thus functions called between fork(2) and exec(2) might deadlock on resources held by threads eliminated in the child process unless these functions are declared as *fork-safe*.

Note: DCE Threads do not declare any function of the thread-safe system libraries to be *fork-safe*.

Fork handlers installed prior to fork(2) via atfork() offer the application control of its own operation across fork(2) . However, these cannot be used to control resources hidden in runtime libraries.

The use of vfork(2) called from a thread is not supported at all.

A call to exit(2) , particularly when executing exit handlers or flushing output streams, might block due to (internal) mutexes held in other threads.

RETURN VALUE

All functions [except pthread_exit(), pthread_getspecific() and pthread_self()] return zero (0) for success and –1 in case of errors. The errno value holds the error code.

FILES

DCE Threads install or modify the following files in the Reliant UNIX files system. The home directory for DCE Threads is /opt/thread with subdirectories bin, include, lib, lib64s, man, readme.

```
INC = ONE OF {pthread.h pthread_exc.h}
INC2 = ONE OF {cma.h cma_config.h cma_errno.h cma_px.h
    cma_sigwait.h cma_ux.h cmalib_crtlx.h exc_handling.h}
LIBCMA = ONE OF {libcma.so libcma_ext.so}
LIBSO_R = ONE OF {libabi_r.so libc_r.so libc_r.so.1 libcwrap.so
    libgen_r.a libnsl_r.so libresolv_r.so libsocket_r.so
    resolv_r.so tcpip_r.so}
LIBA_R = ONE OF {libelf_r.a libgen_r.a libm_r.a libmalloc_r.a}
NO_R_LINKS = ONE OF {libc.so libelf.a libgen.a libm.a
    libmalloc.a libnsl.so libnsl_r_d.so libnsl_r_i.so
    libresolv.so libsocket.so resolv.so tcpip.so}
```

/opt/thread/bin/*
: Executables and scripts

/opt/thread/include/*$INC*
: Include files

/opt/thread/include/dce/*$INC2*
: Include files used indirectly.

/opt/thread/lib/*$LIBCMA*
: API libraries

/opt/thread/lib/*$LIBSO_R*
: Thread-safe system libraries

/opt/thread/lib/*$LIBA_R*
: Thread-safe system archives

/opt/thread/lib/*$NO_R_LINKS*
: Links to files without the "_r" suffix

`/opt/thread/lib64s/`*$LIBCMA*
64-bit API libraries

`/opt/thread/lib64s/`*$LIBSO_R*
64-bit Thread-safe system libraries

`/opt/thread/lib64s/`*$LIBA_R*
64-bit Thread-safe system archives

`/opt/thread/lib/`*$NO_R_LINKS*
Links to libraries without the "_r" suffix

`/opt/thread/readme/*`
Release note

`/etc/netconfig`
DCE Threads add thread-safe libraries *)

There are also links from other places to the files and directories listed above. These are mainly for convenience or migration reasons.

```
LIB_DCE = ONE OF {libabi_r.so libc_r.so libc_r.so.1 libcma.so
    libcwrap.so libelf_r.a libgen_r.a libm_r.a libmalloc_r.a
    libnsl_r.so libnsl_r_d.so libnsl_r_i.so libresolv_r.so
    libsocket_r.so resolv_r.so tcpip_r.so}

LIB_USR = ONE OF {libabi.so libc_r.so.1 libcma.so libcma_ext.so
    libcwrap.so libnsl_r.so libresolv_r.so libsocket_r.so
    resolv_r.so tcpip_r.so}
```

Path	Linked to
`/opt/dcelocal/lib/`*$MOD*	`/opt/thread/lib/`*$MOD*
`/opt/dcelocal/lib/`*$LIB_DCE*	`/opt/thread/lib/`*$LIB_DCE*
`/usr/include/dce/`*$INC*	`/opt/thread/include/`*$INC* *)
`/usr/lib/`*$LIB_USR*	`/opt/thread/lib/`*$LIB_USR* *)
`/usr/lib/`*$LIB_USR*	`/opt/thread/lib64s/`*$LIB_USR* *)

*) These files or links will be reset or lost during a reinstallation of Reliant UNIX.

LITERATURE

The following literature contains specific details and background:

[**DCE Threads**]

OSF DCE Application Guide – Introduction and Style Guide, The Open Group, 1995 *OSF DCE Application Guide – Core Components*, The Open Group, 1995

OSF DCE Application Development Reference, DCE Threads, The Open Group, 1995 *X/Open DCE 1.1: Threads Specification*, X/Open Company Ltd., U.K., 1996

[**POSIX.1**]

Information technology – Portable Operating System Interface (POSIX) – Part 1: System Application Programming Interface (API) [C Language], ISO/IEC 9945-1: 1996, ANSI/IEEE Std. 1003.1, 1996 Edition

[**POSIX.4a/D4**]

Information technology – Portable Operating System Interface (POSIX) – Part 1: System Application Programming Interface (API) – Amendment 2: Threads Extension [C Language], Draft 4, No longer available.

[**POSIX.4a/D10**]
POSIX.1c/D10 Summary, Sun Microsystems, 1995

[**C/C++**]
C/C++ Compiler V1.0 (Reliant UNIX) User Guide, Siemens Nixdorf Informationssysteme AG, Order No. U25438-J-Z145-1-7600, April 1997

SEE ALSO

All manual pages listed by "`apropos pthread`" and the following:

`ctime_r`(3), `getgrgid_r`(3), `gethostent_r`(3), `getpwnam_r`(3), `rand_r`(3), `readdir_r`(3), `strtok_r`(3), `atfork`(3-thr), `DATA_TYPES`(3-thr), `exceptions`(3-thr), `sigaction`(3-thr), `sigpending`(3-thr), `sigprocmask`(3-thr), `sigwait`(3-thr).

atfork(3-thr) - pthread_create(3-thr)

atfork(3-thr)

NAME

`atfork -` arranges for fork cleanup handling

SYNOPSIS

```
#include <pthread.h>
void atfork(□
void *user_state,□
void (*pre_fork)(),□
void (*parent_fork)(),□
void (*child_fork)());□
```

PARAMETERS

user_state
Pointer to the user state that is passed to each routine.

pre_fork
Routine to be called before performing the fork.

parent_fork
Routine to be called in the parent after the fork.

child_fork
Routine to be called in the child after the fork.

DESCRIPTION

The `atfork()` routine allows you to register three routines to be executed at different times relative to a fork. The different times and/or places are as follows:

- Just prior to the fork in the parent process.
- Just after the fork in the parent process.
- Just after the fork in the created (child) process.

Use these routines to clean up just prior to `fork()`, to set up after `fork()`, and to perform locking relative to `fork()`. You are allowed to provide one parameter to be used in conjunction with all the routines. This parameter must be *user_state*.

RETURN VALUES

The `atfork()` routine does not return a value. Instead, an exception is raised if there is insufficient table space to record the handler addresses.

SEE ALSO

fork(2).

Fujitsu Siemens Computers Add-on Reference Manual

DATA_TYPES(3-thr)

NAME

`DATA_TYPES` - data types used by DCE Threads

DESCRIPTION

The DCE Threads data types can be divided into two broad categories: primitive system and application level.

Primitive System Data Types

The first category consists of types that represent structures used by (and internal to) DCE Threads. These types are defined as being primitive system data types.

`pthread_attr_t`
`pthread_cond_t`
`pthread_condattr_t`
`pthread_key_t`
`pthread_mutex_t`
`pthread_mutexattr_t`
`pthread_once_t`
`pthread_t`

Although applications must know about these types, passing them in and receiving them from various DCE Threads routines, the structures themselves are opaque: they cannot be directly modified by applications, and they can be manipulated only (and only in some cases) through specific DCE Threads routines. (The `pthread_key_t` type is somewhat different from the others in this list, in that it is essentially a handle to a thread-private block of memory requested by a call to `pthread_keycreate()`.)

Application Level Data Types

The second category of DCE Threads data consists of types used to describe objects that originate in the application:

`pthread_addr_t`
`pthread_destructor_t`
`pthread_initroutine_t`
`pthread_startroutine_t`
`sigset_t`

All of the above types, with the exception of the last, are various kinds of memory addresses that must be passed by callers of certain DCE Threads routines. These types are extensions to POSIX. They permit DCE Threads to be used on platforms that are not fully ANSI C compliant. While being extensions to permit the use of compilers that are not ANSI C compatible, they are fully portable data types.

The last data type, `sigset_t`, exhibits properties of both primitive system and application level data types. While objects of this type originate in the application, the data type is opaque. A set of functions is provided to manipulate objects of this type.

For further information, see the following descriptions, listed in sorted order.

DATA TYPE DESCRIPTIONS

Following are individual descriptions of each of the DCE Threads data types. The descriptions include the routines where the data type is modified, such as, created, changed or deleted/destroyed, but not the routines referencing or using them that do not change them.

`pthread_addr_t`

A miscellaneous data type representing an address value that must be passed by the caller of various threads routines. Usually the `pthread_addr_t` value is the address of an area which contains various

parameters to be made accessible to an implicitly called routine. For example, when the `pthread_create()` routine is called, one of the parameters passed is a `pthread_addr_t` value that contains an address which will be passed to the start routine which the thread is being created to execute; presumably the routine will extract necessary parameters from the area referenced by this address.

`pthread_attr_t`
: Threads attribute object, used to specify the attributes of a thread when it is created by a call to `pthread_create()`. The object is created by a call to `pthread_attr_create()`, then modified as desired by calls to

  `pthread_attr_setinheritsched()`

  `pthread_attr_setprio()`

  `pthread_attr_setsched()`

  `pthread_attr_setstacksize()`

  (Note that there are `_get` versions of these four calls, which can be used to retrieve the respective values.)

`pthread_cond_t`
: Data type representing a threads condition variable. The variable is created by a call to `pthread_cond_init()`, and destroyed by a call to `pthread_cond_destroy()`.

`pthread_condattr_t`
: Data type representing a threads condition variable attributes object. Created by a call to `pthread_condattr_create()`. The range of possible modifications to a condition variable attributes object is not great: creation [via `pthread_condattr_create()`] and deletion [via `pthread_condattr_delete()`] are all. The object is created with default values.

`pthread_destructor_t`
: Data type, passed in a call to `pthread_keycreate()`, representing the address of a procedure to be called to destroy a data value associated with a unique thread-specific data key value when the thread terminates.

`pthread_initroutine_t`
: Data type representing the address of a procedure that performs a one-time initialization for a thread. It is passed in a call to `pthread_once()`. The `pthread_once()` routine, when called, executes the initialization routine. The specified routine is *guaranteed to be executed only once*, even though the `pthread_once()` call occurs in multithreaded code.

`pthread_key_t`
: Data type representing a thread-specific data key, created by a call to `pthread_keycreate()`. The key is an address of memory. Associating a static block of memory with a specific thread in this way is an alternative to using stack memory for the thread. The key is destroyed by the application-supplied procedure specified by the routine specified using the `pthread_destructor_t` data type in the call to `pthread_keycreate()`.

`pthread_mutex_t`
: Data type representing a mutex object. It is created by a call to `pthread_mutex_init()` and destroyed by a call to `pthread_mutex_destroy()`. Care should be taken not to attempt to destroy a locked object.

`pthread_mutexattr_t`
: Data type representing an attributes object which defines the characteristics of a mutex. Created by a call to `pthread_mutexattr_create()`; modified by calls to `pthread_mutexattr_setkind_np()` (which allows you to specify fast, recursive, or nonrecursive mutexes); passed to `pthread_mutex_init()` to create the mutex with the specified attributes. The only other modification allowed is to destroy the mutex attributes object, with `pthread_mutexattr_delete()`.

`pthread_once_t`
: A data structure that defines the characteristics of the one-time initialization routine executed by calling

`pthread_once()`. The structure is opaque to the application, and cannot be modified by it, but it must be explicitly declared by the client code, and initialized by a call to `pthread_once_init()`. The `pthread_once_t` type must not be an array.

`pthread_startroutine_t`
: Data type representing the address of the application routine or other routine, whatever it is, that a new thread is created to execute as its start routine.

`pthread_t`
: Data type representing a thread handle, created by a call to `pthread_create()`. The thread handle is used thenceforth to identify the thread to calls such as `pthread_cancel()`, `pthread_detach()`, `pthread_equal()` (to which two handles are passed for comparison).

`sigset_t`
: Data type representing a set of signals. It is always an integral or structure type. If a structure, it is intended to be a simple structure, such as, a set of arrays as opposed to a set of pointers. It is opaque in that a set of functions called the `sigsetops` primitives is provided to manipulate signal sets. They operate on signal set data objects addressable by the application, not on any objects known to the system.

The primitives are `sigemptyset()` and `sigfillset()` which initialize the set as either empty or full, `sigaddset()` and `sigdelset()` which add or delete signals from the set, and `sigismember()` which permits the application to check if a object (signal) of type `sigset_t` is a member of the signal set. Applications must call at least one of the initialization primitives at least once for each object of type `sigset_t` prior to any other use of that object (signal set).

The object, or objects, represented by this data type when used by `sigaction()` is (are) used in conjunction with a `sigaction` structure by the `sigaction` function to describe an action to be taken with (a) specified `sigset_t`-type object(s).

Fujitsu Siemens Computers Add-on Reference Manual

exceptions(3-thr)

NAME

`exceptions -` exception handling in DCE Threads

DESCRIPTION

DCE Threads provides the following two ways to obtain information about the status of a threads routine:

- The routine returns a status value to the thread.
- The routine raises an exception.

Before you write a multithreaded program, you must choose only one of the preceding two methods of receiving status. These two methods cannot be used together in the same code module.

The POSIX P1003.4a (pthreads) draft standard specifies that errors be reported to the thread by setting the external variable `errno` to an error code and returning a function value of −1. The threads reference pages document this status value-returning interface. However, an alternative to status values is provided by DCE Threads in the exception-returning interface.

Access to exceptions from the C language is defined by the macros in the `exc_handling.h` file. The `exc_handling.h` header file is included automatically when you include `pthread_exc.h`.

To use the exception-returning interface, replace `#include <pthread.h>` with the following include statement:

```
#include <dce/pthread_exc.h>
```

The following example shows the syntax for handling exceptions:

```
TRY
  try_block
[CATCH (exception_name)
  handler_block]...
[CATCH_ALL
  handler_block]
ENDTRY
```

Fujitsu Siemens Computers Add-on Reference Manual

pthread_attr_create(3-thr)

NAME

`pthread_attr_create -` creates a thread attributes object

SYNOPSIS

```
#include <pthread.h>
int pthread_attr_create(pthread_attr_t *attr);
```

PARAMETERS

attr Thread attributes object created.

DESCRIPTION

The `pthread_attr_create()` routine creates a thread attributes object that is used to specify the attributes of threads when they are created. The attributes object created by this routine is used in calls to `pthread_create()`.

The individual attributes (internal fields) of the attributes object are set to default values. (The default values of each attribute are discussed in the descriptions of the following services.) Use the following routines to change the individual attributes:

`pthread_attr_setinheritsched()`

`pthread_attr_setprio()`

`pthread_attr_setsched()`

`pthread_attr_setstacksize()`

When an attributes object is used to create a thread, the values of the individual attributes determine the characteristics of the new thread. Attributes objects perform in a manner similar to additional parameters. Changing individual attributes does not affect any threads that were previously created using the attributes object.

RETURN VALUES

If the function fails, –1 is returned and `errno` may be set to one of the following values:

`ENOMEM`
: Insufficient memory exists to create the thread attributes object.

`EINVAL`
: The value specified by *attr* is invalid.

SEE ALSO

`pthread_attr_delete`(3-thr), `pthread_attr_setinheritsched`(3-thr), `pthread_attr_setprio`(3-thr), `pthread_attr_setsched`(3-thr), `pthread_attr_setstacksize`(3-thr), `pthread_create`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_attr_delete(3-thr)

NAME

`pthread_attr_delete -` deletes a thread attributes object

SYNOPSIS

```
#include <pthread.h>
int pthread_attr_delete(pthread_attr_t *attr);
```

PARAMETERS

attr Thread attributes object deleted.

DESCRIPTION

The `pthread_attr_delete()` routine deletes a thread attributes object and gives permission to reclaim storage for the thread attributes object. Threads that were created using this thread attributes object are not affected by the deletion of the thread attributes object.

The results of calling this routine are unpredictable if the value specified by the *attr* parameter refers to a thread attributes object that does not exist.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
: The value specified by *attr* is invalid.

SEE ALSO

`pthread_attr_create`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_attr_getinheritsched(3-thr)

NAME

`pthread_attr_getinheritsched -` obtains the inherit schedulingattribute

SYNOPSIS

```
#include <pthread.h>
int pthread_attr_getinheritsched(pthread_attr_t attr);
```

PARAMETERS

attr Thread attributes object whose inherit scheduling attribute is obtained.

DESCRIPTION

The `pthread_attr_getinheritsched()` routine obtains the value of the inherit scheduling attribute in the specified thread attributes object. The inherit scheduling attribute specifies whether threads created using the attributes object inherit the scheduling attributes of the creating thread, or use the scheduling attributes stored in the attributes object that is passed to `pthread_create()`.

The default value of the inherit scheduling attribute is `PTHREAD_INHERIT_SCHED`.

RETURN VALUES

On successful completion, this routine returns the inherit scheduling attribute value.

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
: The value specified by *attr* is invalid.

SEE ALSO

`pthread_attr_create`(3-thr), `pthread_attr_setinheritsched`(3-thr), `pthread_create`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_attr_getprio(3-thr)

NAME

`pthread_attr_getprio -` obtains the scheduling priority attribute

SYNOPSIS

```
#include <pthread.h>
int pthread_attr_getprio(pthread_attr_t attr);
```

PARAMETERS

attr Thread attributes object whose priority attribute is obtained.

DESCRIPTION

The `pthread_attr_getprio()` routine obtains the value of the scheduling priority of threads created using the thread attributes object specified by the *attr* parameter.

RETURN VALUES

On successful completion, this routine returns the scheduling priority attribute value.

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
The value specified by *attr* is invalid.

SEE ALSO

`pthread_attr_create`(3-thr), `pthread_attr_setprio`(3-thr), `pthread_create`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_attr_getsched(3-thr)

NAME

`pthread_attr_getsched -` obtains the value of the scheduling policyattribute

SYNOPSIS

```
#include <pthread.h>
int pthread_attr_getsched(pthread_attr_t attr);
```

PARAMETERS

attr Thread attributes object whose scheduling policy attribute is obtained.

DESCRIPTION

The `pthread_attr_getsched()` routine obtains the scheduling policy of threads created using the thread attributes object specified by the *attr* parameter. The default value of the scheduling attribute is `SCHED_OTHER`.

RETURN VALUES

On successful completion, this routine returns the value of the scheduling policy attribute.

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
The value specified by *attr* is invalid.

SEE ALSO

`pthread_attr_create`(3-thr) , `pthread_attr_setsched`(3-thr) , `pthread_create`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_attr_getstacksize(3-thr)

NAME

`pthread_attr_getstacksize -` obtains the value of the stacksizeattribute

SYNOPSIS

```
#include <pthread.h>
long pthread_attr_getstacksize(pthread_attr_t attr);
```

PARAMETERS

attr Thread attributes object whose stacksize attribute is obtained.

DESCRIPTION

The `pthread_attr_getstacksize()` routine obtains the minimum size (in bytes) of the stack for a thread created using the thread attributes object specified by the *attr* parameter.

RETURN VALUES

On successful completion, this routine returns the stacksize attribute value.

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
The value specified by *attr* is invalid.

SEE ALSO

`pthread_attr_create`(3-thr) , `pthread_attr_setstacksize`(3-thr) , `pthread_create`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_attr_setinheritsched(3-thr)

NAME

`pthread_attr_setinheritsched -` changes the inherit schedulingattribute

SYNOPSIS

```
#include <pthread.h>
int pthread_attr_setinheritsched(
pthread_attr_t attr,
int inherit);
```

PARAMETERS

attr Thread attributes object to be modified.

inherit

New value for the inherit scheduling attribute. Valid values are as follows:

`PTHREAD_INHERIT_SCHED`

This is the default value. The created thread inherits the current priority and scheduling policy of the thread calling `pthread_create()`.

`PTHREAD_DEFAULT_SCHED`

The created thread starts execution with the priority and scheduling policy stored in the thread attributes object.

DESCRIPTION

The `pthread_attr_setinheritsched()` routine changes the inherit scheduling attribute of thread creation. The inherit scheduling attribute specifies whether threads created using the specified thread attributes object inherit the scheduling attributes of the creating thread, or use the scheduling attributes stored in the thread attributes object that is passed to `pthread_create()`.

The first thread in an application that is not created by an explicit call to `pthread_create()` has a scheduling policy of `SCHED_OTHER`. (See the `pthread_attr_setprio()` and `pthread_attr_setsched()` routines for more information on valid priority values and valid scheduling policy values, respectively.)

Inheriting scheduling attributes (instead of using the scheduling attributes stored in the attributes object) is useful when a thread is creating several helper threads – threads that are intended to work closely with the creating thread to cooperatively solve the same problem. For example, inherited scheduling attributes ensure that helper threads created in a sort routine execute with the same priority as the calling thread.

RETURN VALUES

If the function fails, –1 is returned, and `errno` may be set to one of the following values:

`EINVAL`

The value specified by *attr* is invalid.

`EINVAL`

The value specified by *inherit* is invalid.

SEE ALSO

`pthread_attr_create`(3-thr), `pthread_attr_getinheritsched`(3-thr), `pthread_attr_setprio`(3-thr), `pthread_attr_setsched`(3-thr), `pthread_create`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_attr_setprio(3-thr)

NAME

`pthread_attr_setprio -` changes the scheduling priority attribute ofthread creation

SYNOPSIS

```
#include <pthread.h>
int pthread_attr_setprio(
pthread_attr_t *attr,
int priority);
```

PARAMETERS

attr Thread attributes object modified.

priority

New value for the priority attribute. The priority attribute depends on scheduling policy. Valid values fall within one of the following ranges:

`PRI_OTHER_MIN` **<=** *priority* **<=** `PRI_OTHER_MAX`
(use with the `SCHED_OTHER` policy)

`PRI_FIFO_MIN` **<=** *priority* **<=** `PRI_FIFO_MAX`
(use with the `SCHED_FIFO` policy)

`PRI_RR_MIN` **<=** *priority* **<=** `PRI_RR_MAX`
(use with the `SCHED_RR` policy)

`PRI_FG_MIN_NP` **<=** *priority* **<=** `PRI_FG_MAX_NP`
(use with the `SCHED_FG_NP` policy)

`PRI_BG_MIN_NP` **<=** *priority* **<=** `PRI_BG_MAX_NP`
(use with the `SCHED_BG_NP` policy)

The default priority is the midpoint between `PRI_OTHER_MIN` and `PRI_OTHER_MAX`. To specify a minimum or maximum priority, use the appropriate symbol; for example, `PRI_FIFO_MIN` or `PRI_FIFO_MAX`. To specify a value between the minimum and maximum, use an appropriate arithmetic expression. For example, to specify a priority midway between the minimum and maximum for the Round Robin scheduling policy, specify the following concept using your programming language's syntax:

```
pri_rr_mid = (PRI_RR_MIN + PRI_RR_MAX + 1)/2
```

If your expression results in a value outside the range of minimum to maximum, an error results when you attempt to use it.

DESCRIPTION

The `pthread_attr_setprio()` routine sets the execution priority of threads that are created using the attributes object specified by the *attr* parameter.

By default, a created thread inherits the priority of the thread calling `pthread_create()`. To specify a priority using this routine, scheduling inheritance must be disabled at the time the thread is created. Before calling this routine and `pthread_create()`, call `pthread_attr_setinheritsched()` and specify the value `PTHREAD_DEFAULT_SCHED` for the *inherit* parameter.

An application specifies priority only to express the urgency of executing the thread relative to other threads. Priority is not used to control mutual exclusion when accessing shared data.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
The value specified by *attr* is invalid.

ERANGE
One or more parameters supplied have an invalid value.

SEE ALSO

pthread_attr_create(3-thr), pthread_attr_getprio(3-thr), pthread_attr_setinheritsched(3-thr), pthread_create(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_attr_setsched(3-thr)

NAME

`pthread_attr_setsched -` changes the scheduling policy attribute ofthread creation

SYNOPSIS

```
#include <pthread.h>
int pthread_attr_setsched(
pthread_attr_t *attr,
int scheduler);
```

PARAMETERS

attr The thread attributes object modified.

scheduler

The new value for the scheduling policy attribute. Valid values are as follows:

`SCHED_FIFO`
: (First In, First Out) The highest-priority thread runs until it blocks. If there is more than one thread with the same priority, and that priority is the highest among other threads, the first thread to begin running continues until it blocks.

`SCHED_RR`
: (Round Robin) The highest-priority thread runs until it blocks; however, threads of equal priority, if that priority is the highest among other threads, are timesliced. Timeslicing is a process in which threads alternate making use of available processors.

`SCHED_OTHER`
: (Default) All threads are timesliced. `SCHED_OTHER` ensures that all threads, regardless of priority, receive some scheduling so that no thread is completely denied execution time. (However, `SCHED_OTHER` threads can be denied execution time by `SCHED_FIFO` or `SCHED_RR` threads.)

`SCHED_FG_NP`
: (Foreground) Same as `SCHED_OTHER`. Threads are timesliced and priorities can be modified dynamically by the scheduler to ensure fairness.

`SCHED_BG_NP`
: (Background) Ensures that all threads, regardless of priority, receive some scheduling. However, `SCHED_BG_NP` can be denied execution by `SCHED_FIFO` or `SCHED_RR` threads.

DESCRIPTION

The `pthread_attr_setsched()` routine sets the scheduling policy of a thread that is created by using the attributes object specified by the *attr* parameter. The default value of the scheduling attribute is `SCHED_OTHER`.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
: The value specified by *attr* is invalid.

`EINVAL`
: The value specified by *scheduler* is invalid.

`EPERM`
: The caller does not have the appropriate privileges to set the scheduling policy attribute in the specified threads attribute object.

SEE ALSO

`pthread_attr_create`(3-thr), `pthread_attr_getsched`(3-thr), `pthread_attr_setinheritsched`(3-thr), `pthread_create`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_attr_setstacksize(3-thr)

NAME

`pthread_attr_setstacksize -` changes the stacksize attribute ofthread creation

SYNOPSIS

```
#include <pthread.h>
int pthread_attr_setstacksize(
pthread_attr_t *attr,
long stacksize);
```

PARAMETERS

attr Thread attributes object modified.

stacksize

New value for the stacksize attribute. The *stacksize* parameter specifies the minimum size (in bytes) of the stack needed for a thread.

DESCRIPTION

The `pthread_attr_setstacksize()` routine sets the minimum size (in bytes) of the stack needed for a thread created using the attributes object specified by the *attr* parameter. Use this routine to adjust the size of the writable area of the stack. The default value of the stacksize attribute is machine specific.

A thread's stack is fixed at the time of thread creation. Only the main or initial thread can dynamically extend its stack.

Most compilers do not check for stack overflow. Ensure that your thread stack is large enough for anything that you call from the thread.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`

The value specified by *attr* is invalid.

`EINVAL`

The value specified by *stacksize* is invalid.

SEE ALSO

`pthread_attr_create`(3-thr) , `pthread_attr_getstacksize`(3-thr) , `pthread_create`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_cancel(3-thr)

NAME

`pthread_cancel` - allows a thread to request that it or anotherthread terminate execution

SYNOPSIS

```
#include <pthread.h>
int pthread_cancel(pthread_t thread);
```

PARAMETERS

thread
: Thread that receives a cancel request.

DESCRIPTION

The `pthread_cancel()` routine sends a cancel to the specified thread. A cancel is a mechanism by which a calling thread informs either itself or the called thread to terminate as quickly as possible. Issuing a cancel does not guarantee that the canceled thread receives or handles the cancel. The canceled thread can delay processing the cancel after receiving it. For instance, if a cancel arrives during an important operation, the canceled thread can continue if what it is doing cannot be interrupted at the point where the cancel is requested.

Because of communications delays, the calling thread can only rely on the fact that a cancel eventually becomes pending in the designated thread (provided that the thread does not terminate beforehand). Furthermore, the calling thread has no guarantee that a pending cancel is to be delivered because delivery is controlled by the designated thread.

Termination processing when a cancel is delivered to a thread is similar to `pthread_exit()`. Outstanding cleanup routines are executed in the context of the target thread, and a status of −1 is made available to any threads joining with the target thread.

This routine is preferred in implementing Ada's `abort` statement and any other language (or software-defined construct) for requesting thread cancellation.

The results of this routine are unpredictable if the value specified in *thread* refers to a thread that does not currently exist.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
: The specified thread is invalid.

`ERSCH`
: The specified thread does not refer to a currently existing thread.

SEE ALSO

`pthread_exit`(3-thr), `pthread_join`(3-thr), `pthread_setasynccancel`(3-thr), `pthread_setcancel`(3-thr), `pthread_testcancel`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_cleanup_pop(3-thr)

NAME

`pthread_cleanup_pop -` removes the cleanup handler at the top of thecleanup stack and optionally executes it

SYNOPSIS

```
#include <pthread.h>
void pthread_cleanup_pop(int execute);
```

PARAMETERS

execute

Integer that specifies whether the cleanup routine that is popped should be executed or just discarded. If the value is nonzero, the cleanup routine is executed.

DESCRIPTION

The `pthread_cleanup_pop()` routine removes the routine specified in `pthread_cleanup_push()` from the top of the calling thread's cleanup stack and executes it if the value specified in *execute* is nonzero.

This routine and `pthread_cleanup_push()` are implemented as macros and must be displayed as statements and in pairs within the same lexical scope. You can think of the `pthread_cleanup_push()` macro as expanding to a string whose first character is a { (left brace) and `pthread_cleanup_pop` as expanding to a string containing the corresponding } (right brace).

RETURN VALUES

This routine must be used as a statement.

SEE ALSO

`pthread_cleanup_push`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_cleanup_push(3-thr)

NAME

`pthread_cleanup_push -` establishes a cleanup handler

SYNOPSIS

```
#include <pthread.h>
void pthread_cleanup_push(
void routine,
pthread_addr_t arg);
```

PARAMETERS

routine
: Routine executed as the cleanup handler.

arg
: Parameter executed with the cleanup routine.

DESCRIPTION

The `pthread_cleanup_push()` routine pushes the specified routine onto the calling thread's cleanup stack. The cleanup routine is popped from the stack and executed with the *arg* parameter when any of the following actions occur:

- The thread calls `pthread_exit()`.
- The thread is canceled.
- The thread calls `pthread_cleanup_pop()` and specifies a nonzero value for the *execute* parameter.

This routine and `pthread_cleanup_pop()` are implemented as macros and must be displayed as statements and in pairs within the same lexical scope. You can think of the `pthread_cleanup_push()` macro as expanding to a string whose first character is a { (left brace) and `pthread_cleanup_pop()` as expanding to a string containing the corresponding } (right brace).

RETURN VALUES

This routine must be used as a statement.

SEE ALSO

`pthread_cancel`(3-thr), `pthread_cleanup_pop`(3-thr), `pthread_exit`(3-thr), `pthread_testcancel`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_cond_broadcast(3-thr)

NAME

`pthread_cond_broadcast -` wakes all threads that are waiting on acondition variable

SYNOPSIS

```
#include <pthread.h>
int pthread_cond_broadcast(pthread_cond_t *cond);
```

PARAMETERS

cond Condition variable broadcast.

DESCRIPTION

The `pthread_cond_broadcast()` routine wakes all threads waiting on a condition variable. Calling this routine implies that data guarded by the associated mutex has changed so that it might be possible for one or more waiting threads to proceed. If any one waiting thread might be able to proceed, call `pthread_cond_signal()`.

Call this routine when the associated mutex is either locked or unlocked.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
: The value specified by *cond* is invalid.

SEE ALSO

`pthread_cond_destroy`(3-thr), `pthread_cond_init`(3-thr), `pthread_cond_signal`(3-thr), `pthread_cond_timedwait`(3-thr), `pthread_cond_wait`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_cond_destroy(3-thr)

NAME

`pthread_cond_destroy` - deletes a condition variable

SYNOPSIS

```
#include <pthread.h>
int pthread_cond_destroy(pthread_cond_t *cond);
```

PARAMETERS

cond Condition variable deleted.

DESCRIPTION

The `pthread_cond_destroy()` routine deletes a condition variable. Call this routine when a condition variable is no longer referenced. The effect of calling this routine is to give permission to reclaim storage for the condition variable.

The results of this routine are unpredictable if the condition variable specified in *cond* does not exist.

The results of this routine are also unpredictable if there are threads waiting for the specified condition variable to be signaled or broadcast when it is deleted.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
: The value specified by *cond* is invalid.

`EBUSY`
: A thread is currently executing a `pthread_cond_timedwait()` routine or `pthread_cond_wait()` on the condition variable specified in *cond*.

SEE ALSO

`pthread_cond_broadcast`(3-thr), `pthread_cond_init`(3-thr), `pthread_cond_signal`(3-thr), `pthread_cond_timedwait`(3-thr), `pthread_cond_wait`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_cond_init(3-thr)

NAME

`pthread_cond_init -` creates a condition variable

SYNOPSIS

```
#include <pthread.h>
int pthread_cond_init(
pthread_cond_t *cond,
pthread_condattr_t attr);
```

PARAMETERS

cond Condition variable that is created.

attr Condition variable attributes object that defines the characteristics of the condition variable created. If you specify `pthread_condattr_default`, default attributes are used.

DESCRIPTION

The `pthread_cond_init()` routine creates and initializes a condition variable. A condition variable is a synchronization object used in conjunction with a mutex. A mutex controls access to shared data; a condition variable allows threads to wait for that data to enter a defined state. The state is defined by a Boolean expression called a predicate.

A condition variable is signaled or broadcast to indicate that a predicate might have become true. The broadcast operation indicates that all waiting threads need to resume and reevaluate the predicate. The signal operation is used when any one waiting thread can continue.

If a thread that holds a mutex determines that the shared data is not in the correct state for it to proceed (the associated predicate is not true), it waits on a condition variable associated with the desired state. Waiting on the condition variable automatically releases the mutex so that other threads can modify or examine the shared data. When a thread modifies the state of the shared data so that a predicate might be true, it signals or broadcasts on the appropriate condition variable so that threads waiting for that predicate can continue.

It is important that all threads waiting on a particular condition variable at any time hold the *same* mutex. If they do not, the behavior of the wait operation is unpredictable (an implementation can use the mutex to control internal access to the condition variable object). However, it is legal for a client to store condition variables and mutexes and later reuse them in different combinations. The client must ensure that no threads use the condition variable with the old mutex. At any time, an arbitrary number of condition variables can be associated with a single mutex, each representing a different predicate of the shared data protected by that mutex.

Condition variables are not owned by a particular thread. Any associated storage is not automatically deallocated when the creating thread terminates.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EAGAIN`
The system lacks the necessary resources to initialize another condition variable.

`EINVAL`
Invalid attributes object.

`ENOMEM`
Insufficient memory exists to initialize the condition variable.

SEE ALSO

`pthread_cond_broadcast`(3-thr) , `pthread_cond_destroy`(3-thr) , `pthread_cond_signal`(3-thr) ,

`pthread_cond_timedwait`(3-thr), `pthread_cond_wait`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_cond_signal(3-thr)

NAME

`pthread_cond_signal -` wakes one thread that is waiting on acondition variable

SYNOPSIS

```
#include <pthread.h>
int pthread_cond_signal(pthread_cond_t *cond);
```

PARAMETERS

cond Condition variable signaled.

DESCRIPTION

The `pthread_cond_signal()` routine wakes one thread waiting on a condition variable. Calling this routine implies that data guarded by the associated mutex has changed so that it is possible for a single waiting thread to proceed. Call this routine when any thread waiting on the specified condition variable might find its predicate true, but only one thread needs to proceed.

The scheduling policy determines which thread is awakened. For policies `SCHED_FIFO` and `SCHED_RR` a blocked thread is chosen in priority order.

Call this routine when the associated mutex is either locked or unlocked.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
: The value specified by *cond* is invalid.

SEE ALSO

`pthread_cond_broadcast`(3-thr), `pthread_cond_destroy`(3-thr), `pthread_cond_init`(3-thr), `pthread_cond_timedwait`(3-thr), `pthread_cond_wait`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_cond_timedwait(3-thr)

NAME

`pthread_cond_timedwait -` causes a thread to wait for a conditionvariable to be signaled or broadcast

SYNOPSIS

```
#include <pthread.h>
int pthread_cond_timedwait(
pthread_cond_t *cond,
pthread_mutex_t *mutex,
struct timespec *abstime);
```

PARAMETERS

cond Condition variable waited on.

mutex
Mutex associated with the condition variable specified in *cond*.

abstime
Absolute time at which the wait expires, if the condition has not been signaled or broadcast. (See the `pthread_get_expiration_np()` routine, which you can use to obtain a value for this parameter.)

DESCRIPTION

The `pthread_cond_timedwait()` routine causes a thread to wait until one of the following occurs:

- The specified condition variable is signaled or broadcast.
- The current system clock time is greater than or equal to the time specified by the *abstime* parameter.

This routine is identical to `pthread_cond_wait()` except that this routine can return before a condition variable is signaled or broadcast – specifically, when a specified time expires.

If the current time equals or exceeds the expiration time, this routine returns immediately, without causing the current thread to wait.

Call this routine after you lock the mutex specified in *mutex*. The results of this routine are unpredictable if this routine is called without first locking the mutex.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
The value specified by *cond*, *mutex*, or *abstime* is invalid.

`EAGAIN`
The time specified by *abstime* expired.

`EDEADLK`
A deadlock condition is detected.

SEE ALSO

`pthread_cond_broadcast`(3-thr), `pthread_cond_destroy`(3-thr), `pthread_cond_init`(3-thr), `pthread_cond_signal`(3-thr), `pthread_cond_wait`(3-thr), `pthread_get_expiration_np`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_cond_wait(3-thr)

NAME

`pthread_cond_wait -` causes a thread to wait for a condition variableto be signaled or broadcast

SYNOPSIS

```
#include <pthread.h>
int pthread_cond_wait(
pthread_cond_t *cond,
pthread_mutex_t *mutex);
```

PARAMETERS

cond Condition variable waited on.

mutex
Mutex associated with the condition variable specified in *cond*.

DESCRIPTION

The `pthread_cond_wait()` routine causes a thread to wait for a condition variable to be signaled or broadcast. Each condition corresponds to one or more predicates based on shared data. The calling thread waits for the data to reach a particular state (for the predicate to become true).

Call this routine after you have locked the mutex specified in *mutex*. The results of this routine are unpredictable if this routine is called without first locking the mutex.

This routine automatically releases the mutex and causes the calling thread to wait on the condition. If the wait is satisfied as a result of some thread calling `pthread_cond_signal()` or `pthread_cond_broadcast()`, the mutex is reacquired and the routine returns.

A thread that changes the state of storage protected by the mutex in such a way that a predicate associated with a condition variable might now be true must call either `pthread_cond_signal()` or `pthread_cond_broadcast()` for that condition variable. If neither call is made, any thread waiting on the condition variable continues to wait.

This routine might (with low probability) return when the condition variable has not been signaled or broadcast. When a spurious wakeup occurs, the mutex is reacquired before the routine returns. (To handle this type of situation, enclose this routine in a loop that checks the predicate.)

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
The value specified by *cond* or *mutex* is invalid.

`EDEADLK`
A deadlock condition is detected.

SEE ALSO

`pthread_cond_broadcast`(3-thr), `pthread_cond_destroy`(3-thr), `pthread_cond_init`(3-thr), `pthread_cond_signal`(3-thr), `pthread_cond_timedwait`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_condattr_create(3-thr)

NAME

`pthread_condattr_create -` creates a condition variable attributesobject

SYNOPSIS

```
#include <pthread.h>
int pthread_condattr_create(pthread_condattr_t *attr);
```

PARAMETERS

attr Condition variable attributes object that is created.

DESCRIPTION

The `pthread_condattr_create()` routine creates a condition variable attributes object that is used to specify the attributes of condition variables when they are created. The condition variable attributes object is initialized with the default value for all of the attributes defined by a given implementation.

When a condition variable attributes object is used to create a condition variable, the values of the individual attributes determine the characteristics of the new object. Attributes objects act like additional parameters to object creation. Changing individual attributes does not affect objects that were previously created using the attributes object.

RETURN VALUES

The created condition variable attributes object is returned to the *attr* parameter.

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
: The value specified by *attr* is invalid.

`ENOMEM`
: Insufficient memory exists to create the condition variable attributes object.

SEE ALSO

`pthread_cond_init`(3-thr), `pthread_condattr_delete`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_condattr_delete(3-thr)

NAME

`pthread_condattr_delete -` deletes a condition variable attributesobject

SYNOPSIS

```
#include <pthread.h>
int pthread_condattr_delete(pthread_condattr_t *attr);
```

PARAMETERS

attr Condition variable attributes object deleted.

DESCRIPTION

The `pthread_condattr_delete()` routine deletes a condition variable attributes object. Call this routine when a condition variable attributes object created by `pthread_condattr_create()` is no longer referenced.

This routine gives permission to reclaim storage for the condition variable attributes object. Condition variables that are created using this attributes object are not affected by the deletion of the condition variable attributes object.

The results of calling this routine are unpredictable if the handle specified by the *attr* parameter refers to an attributes object that does not exist.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
: The value specified by *attr* is invalid.

SEE ALSO

`pthread_condattr_create`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_create(3-thr)

NAME

`pthread_create -` creates a thread object and thread

SYNOPSIS

```
#include <pthread.h>
int pthread_create(
pthread_t *thread,
pthread_attr_t attr,
pthread_startroutine_t start_routine,
pthread_addr_t arg);
```

PARAMETERS

thread
: Handle to the thread object created.

attr
: Thread attributes object that defines the characteristics of the thread being created. If you specify `pthread_attr_default`, default attributes are used.

start_routine
: Function executed as the new thread's start routine.

arg
: Address value copied and passed to the thread's start routine.

DESCRIPTION

The `pthread_create()` routine creates a thread object and a thread. A thread is a single, sequential flow of control within a program. It is the active execution of a designated routine, including any nested routine invocations. A thread object defines and controls the executing thread.

Creating a Thread

Calling this routine sets into motion the following actions:

- An internal thread object is created to describe the thread.
- The associated executable thread is created with attributes specified by the *attr* parameter (or with default attributes if `pthread_attr_default` is specified).
- The *thread* parameter receives the new thread.
- The *start_routine* function is called. This may occur before this routine returns successfully.

Thread Execution

The thread is created in the ready state and therefore might immediately begin executing the function specified by the *start_routine* parameter. The newly created thread begins running before `pthread_create()` completes if the new thread follows the `SCHED_RR` or `SCHED_FIFO` scheduling policy or has a priority higher than the creating thread, or both. Otherwise, the new thread begins running at its turn, which with sufficient processors might also be before `pthread_create()` returns.

The *start_routine* parameter is passed a copy of the *arg* parameter. The value of the *arg* parameter is unspecified.

The thread object exists until the `pthread_detach()` routine is called or the thread terminates, whichever occurs last.

The synchronization between the caller of `pthread_create()` and the newly created thread is through the use of the `pthread_join()` routine (or any other mutexes or condition variables they agree to use).

Terminating a Thread

A thread terminates when one of the following events occurs:

- The thread returns from its start routine.
- The thread exits (within a routine) as the result of calling the `pthread_exit()` routine.
- The thread is canceled.

When a Thread Terminates

The following actions are performed when a thread terminates:

- If the thread terminates by returning from its start routine or calling `pthread_exit()`, the return value is copied into the thread object. If the start routine returns normally and the start routine is a procedure that does not return a value, then the result obtained by `pthread_join()` is unpredictable. If the thread has been cancelled, a return value of –1 is copied into the thread object. The return value can be retrieved by other threads by calling the `pthread_join()` routine.
- A destructor for each thread-specific data point is removed from the list of destructors for this thread and then is called. This step destroys all the thread-specific data associated with the current thread.
- Each cleanup handler that has been declared by `pthread_cleanup_push()` and not yet removed by `pthread_cleanup_pop()` is called. The most recently pushed handler is called first.
- A flag is set in the thread object indicating that the thread has terminated. This flag must be set in order for callers of `pthread_join()` to return from the call.
- A broadcast is made so that all threads currently waiting in a call to `pthread_join()` can return from the call.
- The thread object is marked to indicate that it is no longer needed by the thread itself. A check is made to determine if the thread object is no longer needed by other threads; that is, if `pthread_detach()` has been called. If that routine is called, then the thread object is deallocated.

RETURN VALUES

Upon successful completion, this routine stores the identifier of the created thread at *thread* and returns 0. Otherwise, a value of –1 is returned and no thread is created, the contents of *thread* are undefined, and `errno` may be set to one of the following values:

`EAGAIN`
: The system lacks the necessary resources to create another thread.

`ENOMEM`
: Insufficient memory exists to create the thread object. This is not a temporary condition.

SEE ALSO

`pthread_attr_create`(3-thr), `pthread_cancel`(3-thr), `pthread_detach`(3-thr), `pthread_exit`(3-thr), `pthread_join`(3-thr).

pthread_delay_np(3-thr) - pthread_once(3-thr)

pthread_delay_np(3-thr)

NAME

`pthread_delay_np` - causes a thread to wait for a specified period

SYNOPSIS

```
#include <pthread.h>
int pthread_delay_np(struct timespec *interval);
```

PARAMETERS

interval

Number of seconds and nanoseconds that the calling thread waits before continuing execution. The value specified must be greater than or equal to 0 (zero).

DESCRIPTION

The `pthread_delay_np()` routine causes a thread to delay execution for a specified period of elapsed wall clock time. The period of time the thread waits is at least as long as the number of seconds and nanoseconds specified in the *interval* parameter.

Specifying an interval of 0 (zero) seconds and 0 (zero) nanoseconds is allowed and can result in the thread giving up the processor or delivering a pending cancel.

The `struct timespec` structure contains two fields, as follows:

- The `tv_sec` field is an integer number of seconds.
- The `tv_nsec` field is an integer number of nanoseconds.

This routine is a new primitive.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
: The value specified by *interval* is invalid.

SEE ALSO

`pthread_yield`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_detach(3-thr)

NAME

`pthread_detach -` marks a thread object for deletion

SYNOPSIS

```
#include <pthread.h>
int pthread_detach(pthread_t *thread);
```

PARAMETERS

thread
Thread object marked for deletion.

DESCRIPTION

The `pthread_detach()` routine indicates that storage for the specified thread is reclaimed when the thread terminates. This includes storage for the *thread* parameter's return value. If *thread* has not terminated when this routine is called, this routine does not cause it to terminate.

Call this routine when a thread object is no longer referenced. Additionally, call this routine for every thread that is created to ensure that storage for thread objects does not accumulate.

You cannot join with a thread after the thread has been detached.

The results of this routine are unpredictable if the value of *thread* refers to a thread object that does not exist.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
The value specified by *thread* is invalid.

`ESRCH`
The value specified by *thread* does not refer to an existing thread.

SEE ALSO

`pthread_cancel`(3-thr), `pthread_create`(3-thr), `pthread_exit`(3-thr), `pthread_join`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_equal(3-thr)

NAME

`pthread_equal` - compares one thread identifier to another threadidentifier

SYNOPSIS

```
#include <pthread.h>
boolean32 pthread_equal(
pthread_t *thread1,
pthread_t *thread2);
```

PARAMETERS

thread1
: The first thread identifier to be compared.

thread2
: The second thread identifier to be compared.

DESCRIPTION

This routine compares one thread identifier to another thread identifier. (This routine does not check whether the objects that correspond to the identifiers currently exist.) If the identifiers have values indicating that they designate the same object, 1 (true) is returned. If the values do not designate the same object, 0 (false) is returned.

This routine is implemented as a C macro.

RETURN VALUES

Possible return values are as follows:

`0` Values of *thread1* and *thread2* do not designate the same object.

`1` Values of *thread1* and *thread2* designate the same object.

SEE ALSO

`pthread_create`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_exit(3-thr)

NAME

`pthread_exit` - terminates the calling thread

SYNOPSIS

```
#include <pthread.h>
void pthread_exit(pthread_addr_t status);
```

PARAMETERS

status

Address value copied and returned to the caller of `pthread_join()`.

DESCRIPTION

The `pthread_exit()` routine terminates the calling thread and makes a status value available to any thread that calls `pthread_join()` and specifies the terminating thread.

An implicit call to `pthread_exit()` is issued when a thread returns from the start routine that was used to create it. The function's return value serves as the thread's exit status. If the return value is −1, an error exit is forced for the thread instead of a normal exit. The process exits when the last running thread calls `pthread_exit()`, with an undefined exit status.

Restrictions

The `pthread_exit()` routine does not work in the main (initial) thread because DCE Threads relies on information at the base of thread stacks; this information does not exist in the main thread.

RETURN VALUES

No value is returned.

SEE ALSO

`pthread_create`(3-thr), `pthread_detach`(3-thr), `pthread_join`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_get_expiration_np(3-thr)

NAME

`pthread_get_expiration_np -` obtains a value representing a desiredexpiration time

SYNOPSIS

```
#include <pthread.h>
int pthread_get_expiration_np(
struct timespec *delta,
struct timespec *abstime);
```

PARAMETERS

delta

Number of seconds and nanoseconds to add to the current system time. The result is the time when a timed wait expires.

abstime

Value representing the expiration time.

DESCRIPTION

The `pthread_get_expiration_np()` routine adds a specified interval to the current absolute system time and returns a new absolute time. This new absolute time is used as the expiration time in a call to `pthread_cond_timedwait()`. This routine is a new primitive.

The `struct timespec` structure contains two fields, as follows:

- The `tv_sec` field is an integer number of seconds.
- The `tv_nsec` field is an integer number of nanoseconds.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`

The value specified by *delta* is invalid.

SEE ALSO

`pthread_cond_timedwait`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_getprio(3-thr)

NAME

`pthread_getprio` - obtains the current priority of a thread

SYNOPSIS

```
#include <pthread.h>
int pthread_getprio(pthread_t thread);
```

PARAMETERS

thread
Thread whose priority is obtained.

DESCRIPTION

The `pthread_getprio()` routine obtains the current priority of a thread. The current priority is different from the initial priority of the thread if the `pthread_setprio()` routine is called.

The exact effect of different priority values depends upon the scheduling policy assigned to the thread.

RETURN VALUES

The current priority value of the thread specified in *thread* is returned. (See the `pthread_setprio()` reference page for valid values.)

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
The value specified by *thread* is invalid.

`ESRCH`
The value specified by *thread* does not refer to an existing thread.

SEE ALSO

`pthread_attr_setprio`(3-thr), `pthread_setprio`(3-thr), `pthread_setscheduler`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_getscheduler(3-thr)

NAME

`pthread_getscheduler -` obtains the current scheduling policy of athread

SYNOPSIS

```
#include <pthread.h>
int pthread_getscheduler(pthread_t thread);
```

PARAMETERS

thread

Thread whose scheduling policy is obtained.

DESCRIPTION

The `pthread_getscheduler()` routine obtains the current scheduling policy of a thread. The current scheduling policy of a thread is different from the initial scheduling policy if the `pthread_setscheduler()` routine is called.

RETURN VALUES

The current scheduling policy value of the thread specified in *thread* is returned. (See the `pthread_setscheduler()` reference page for valid values.)

If the function fails, `errno` may be set to one of the following values:

`EINVAL`

The value specified by *thread* is invalid.

`ESRCH`

The value specified by *thread* does not refer to an existing thread.

SEE ALSO

`pthread_attr_setscheduler`(3-thr), `pthread_setscheduler`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_getspecific(3-thr)

NAME

`pthread_getspecific -` obtains the thread-specific data associatedwith the specified key

SYNOPSIS

```
#include <pthread.h>
int pthread_getspecific(
pthread_key_t key,
pthread_addr_t *value);
```

PARAMETERS

key Context key value that identifies the data value obtained. This key value must be obtained from `pthread_keycreate()`.

value
Address of the current thread-specific data value associated with the specified key.

DESCRIPTION

The `pthread_getspecific()` routine obtains the thread-specific data associated with the specified key for the current thread.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
The key value is invalid.

SEE ALSO

`pthread_keycreate`(3-thr) , `pthread_setspecific`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_join(3-thr)

NAME

`pthread_join -` causes the calling thread to wait for the terminationof a specified thread

SYNOPSIS

```
#include <pthread.h>
int pthread_join(
pthread_t thread,
pthread_addr_t *status);
```

PARAMETERS

thread
: Thread whose termination is awaited by the caller of this routine.

status
: Status value of the terminating thread when that thread calls `pthread_exit()`.

DESCRIPTION

The `pthread_join()` routine causes the calling thread to wait for the termination of a specified thread. A call to this routine returns after the specified thread has terminated.

Any number of threads can call this routine. All threads are awakened when the specified thread terminates.

If the current thread calls this routine to join with itself, an error is returned.

The results of this routine are unpredictable if the value for *thread* refers to a thread object that no longer exists.

RETURN VALUES

If the thread terminates normally, the exit status is the value that is optionally returned from the thread's start routine.

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
: The value specified by *thread* is invalid.

`ESRCH`
: The value specified by *thread* does not refer to a currently existing thread.

`EDEADLK`
: A deadlock is detected.

SEE ALSO

`pthread_create`(3-thr), `pthread_detach`(3-thr), `pthread_exit`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_keycreate(3-thr)

NAME

`pthread_keycreate -` generates a unique thread-specific data keyvalue

SYNOPSIS

```
#include <pthread.h>
int pthread_keycreate(
pthread_key_t *key,
void (*destructor) (void *value));
```

PARAMETERS

key Value of the new thread-specific data key.

destructor
Procedure to be called to destroy a data value associated with the created key when the thread terminates.

DESCRIPTION

The `pthread_keycreate()` routine generates a unique thread-specific data key value. This key value identifies a thread-specific data value, which is an address of memory generated by the client containing arbitrary data of any size.

Thread-specific data allows client software to associate information with the current thread.

For example, thread-specific data can be used by a language runtime library that needs to associate a language-specific thread-private data structure with an individual thread. The thread-specific data routines also provide a portable means of implementing the class of storage called thread-private static, which is needed to support parallel decomposition in the FORTRAN language.

This routine generates and returns a new key value. Each call to this routine within a process returns a key value that is unique within an application invocation. Calls to `pthread_keycreate()` must occur in initialization code guaranteed to execute only once in each process. The `pthread_once()` routine provides a way of specifying such code.

When multiple facilities share access to thread-specific data, the facilities must agree on the key value that is associated with the context. The key value must be created only once and needs to be stored in a location known to each facility. (It may be desirable to encapsulate the creation of a key, and the setting and getting of context values for that key, within a special facility created for that purpose.)

When a thread terminates, thread-specific data is automatically destroyed. For each thread-specific data currently associated with the thread, the *destructor* routine associated with the key value of that context is called. The order in which per-thread context destructors are called at thread termination is undefined.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
The value specified by *key* is invalid.

`EAGAIN`
An attempt was made to allocate a key when the key name space is exhausted. This is not a temporary condition.

`ENOMEM`
Insufficient memory exists to create the key.

SEE ALSO

`pthread_getspecific`(3-thr) , `pthread_setspecific`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_lock_global_np(3-thr)

NAME

`pthread_lock_global_np -` locks the global mutex

SYNOPSIS

```
#include <pthread.h>
void pthread_lock_global_np();
```

DESCRIPTION

The `pthread_lock_global_np()` routine locks the global mutex. If the global mutex is currently held by another thread when a thread calls this routine, the thread waits for the global mutex to become available.

The thread that has locked the global mutex becomes its current owner and remains the owner until the same thread has unlocked it. This routine returns with the global mutex in the locked state and with the current thread as the global mutex's current owner.

Use the global mutex when calling a library package that is not designed to run in a multithreaded environment. (Unless the documentation for a library function specifically states that it is compatible with multithreading, assume that it is not compatible; in other words, assume it is nonreentrant.)

The global mutex is one lock. Any code that calls any function that is not known to be reentrant uses the same lock. This prevents dependencies among threads calling library functions and those functions calling other functions, and so on.

The global mutex is a recursive mutex. A thread that has locked the global mutex can relock it without deadlocking. (The locking thread must call `pthread_unlock_global_np()` as many times as it called this routine to allow another thread to lock the global mutex.)

This routine is a new primitive.

RETURN VALUES

No value is returned.

SEE ALSO

`pthread_mutex_lock`(3-thr) , `pthread_mutex_unlock`(3-thr) , `pthread_mutexattr_setkind_np`(3-thr) , `pthread_unlock_global_np`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_mutex_destroy(3-thr)

NAME

`pthread_mutex_destroy -` deletes a mutex

SYNOPSIS

```
#include <pthread.h>
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

PARAMETERS

mutex

Mutex to be deleted.

DESCRIPTION

The `pthread_mutex_destroy()` routine deletes a mutex and must be called when a mutex object is no longer referenced. The effect of calling this routine is to reclaim storage for the mutex object.

It is illegal to delete a mutex that has a current owner (in other words, is locked).

The results of this routine are unpredictable if the mutex object specified in the *mutex* parameter does not currently exist.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EBUSY`

An attempt was made to destroy a mutex that is locked.

`EINVAL`

The value specified by *mutex* is invalid.

SEE ALSO

`pthread_mutex_init`(3-thr) , `pthread_mutex_lock`(3-thr) , `pthread_mutex_trylock`(3-thr) , `pthread_mutex_unlock`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_mutex_init(3-thr)

NAME

`pthread_mutex_init -` creates a mutex

SYNOPSIS

```
#include <pthread.h>
int pthread_mutex_init(
pthread_mutex_t *mutex,
pthread_mutexattr_t attr);
```

PARAMETERS

mutex
: Mutex that is created.

attr
: Attributes object that defines the characteristics of the created mutex. If you specify `pthread_mutexattr_default`, default attributes are used.

DESCRIPTION

The `pthread_mutex_init()` routine creates a mutex and initializes it to the unlocked state. If the thread that called this routine terminates, the created mutex is not automatically deallocated, because it is considered shared among multiple threads.

RETURN VALUES

If an error condition occurs, this routine returns −1, the mutex is not initialized, the contents of *mutex* are undefined, and `errno` may be set to one of the following values:

`EAGAIN`
: The system lacks the necessary resources to initialize another mutex.

`EINVAL`
: The value specified by *attr* is invalid.

`ENOMEM`
: Insufficient memory exists to initialize the mutex.

SEE ALSO

`pthread_mutex_lock`(3-thr) , `pthread_mutex_trylock`(3-thr) , `pthread_mutex_unlock`(3-thr) , `pthread_mutexattr_create`(3-thr) , `pthread_mutexattr_getkind_np`(3-thr) , `pthread_mutexattr_setkind_np`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_mutex_lock(3-thr)

NAME

`pthread_mutex_lock -` locks an unlocked mutex

SYNOPSIS

```
#include <pthread.h>
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

PARAMETERS

mutex
Mutex that is locked.

DESCRIPTION

The `pthread_mutex_lock()` routine locks a mutex. If the mutex is locked when a thread calls this routine, the thread waits for the mutex to become available.

The thread that has locked a mutex becomes its current owner and remains the owner until the same thread has unlocked it. This routine returns with the mutex in the locked state and with the current thread as the mutex's current owner.

If you specified a fast mutex in a call to `pthread_mutexattr_setkind_np()`, a deadlock can result if the current owner of a mutex calls this routine in an attempt to lock the mutex a second time. If you specified a recursive mutex in a call to `pthread_mutexattr_setkind_np()`, the current owner of a mutex can relock the same mutex without blocking. If you specify a nonrecursive mutex in a call to `pthread_mutexattr_setkind_np()`, an error is returned and the thread does not block if the current owner of a mutex calls this routine in an attempt to lock the mutex a second time.

The preemption of a lower-priority thread that locks a mutex possibly results in the indefinite blocking of higher-priority threads waiting for the same mutex. The execution of the waiting higher-priority threads is blocked for as long as there is a sufficient number of runnable threads of any priority between the lower-priority and higher-priority values. Priority inversion occurs when any resource is shared between threads with different priorities.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
The value specified by *mutex* is invalid.

`EDEADLK`
A deadlock condition is detected.

SEE ALSO

`pthread_mutex_destroy`(3-thr), `pthread_mutex_init`(3-thr), `pthread_mutex_trylock`(3-thr), `pthread_mutex_unlock`(3-thr), `pthread_mutexattr_setkind_np`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_mutex_trylock(3-thr)

NAME

`pthread_mutex_trylock -` locks a mutex

SYNOPSIS

```
#include <pthread.h>
int pthread_mutex_trylock(pthread_mutex_t *mutex);
```

PARAMETERS

mutex
Mutex that is locked.

DESCRIPTION

The `pthread_mutex_trylock()` routine locks a mutex. If the specified mutex is locked when a thread calls this routine, the calling thread does not wait for the mutex to become available.

When a thread calls this routine, an attempt is made to lock the mutex immediately. If the mutex is successfully locked, 1 is returned and the current thread is then the mutex's current owner.

If the mutex is locked by another thread when this routine is called, 0 (zero) is returned and the thread does not wait to acquire the lock. If a fast mutex is owned by the current thread, 0 is returned. If a recursive mutex is owned by the current thread, 1 is returned and the mutex is relocked. (To unlock a recursive mutex, each call to `pthread_mutex_trylock()` must be matched by a call to the `pthread_mutex_unlock()` routine.)

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
The value specified by *mutex* is invalid.

SEE ALSO

`pthread_mutex_destroy`(3-thr), `pthread_mutex_init`(3-thr), `pthread_mutex_lock`(3-thr), `pthread_mutex_unlock`(3-thr), `pthread_mutexattr_setkind_np`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_mutex_unlock(3-thr)

NAME

`pthread_mutex_unlock` - unlocks a mutex

SYNOPSIS

```
#include <pthread.h>
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

PARAMETERS

mutex
Mutex that is unlocked.

DESCRIPTION

The `pthread_mutex_unlock()` routine unlocks a mutex. If no threads are waiting for the mutex, the mutex unlocks with no current owner. If one or more threads are waiting to lock the specified mutex, this routine causes one thread to return from its call to `pthread_mutex_lock()`. The scheduling policy is used to determine which thread acquires the mutex. For the `SCHED_FIFO` and `SCHED_RR` policies, a blocked thread is chosen in priority order.

The results of calling this routine are unpredictable if the mutex specified in *mutex* is unlocked. The results of calling this routine are also unpredictable if the mutex specified in *mutex* is currently owned by a thread other than the calling thread.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
The value specified by *mutex* is invalid.

SEE ALSO

`pthread_mutex_destroy`(3-thr), `pthread_mutex_init`(3-thr), `pthread_mutex_lock`(3-thr), `pthread_mutex_trylock`(3-thr), `pthread_unlock_global_np`(3-thr), `pthread_mutexattr_setkind_np`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_mutexattr_create(3-thr)

NAME

`pthread_mutexattr_create -` creates a mutex attributes object

SYNOPSIS

```
#include <pthread.h>
int pthread_mutexattr_create(pthread_mutexattr_t *attr);
```

PARAMETERS

attr Mutex attributes object created.

DESCRIPTION

The `pthread_mutexattr_create()` routine creates a mutex attributes object used to specify the attributes of mutexes when they are created. The mutex attributes object is initialized with the default value for all of the attributes defined by a given implementation.

When a mutex attributes object is used to create a mutex, the values of the individual attributes determine the characteristics of the new object. Attributes objects act like additional parameters to object creation. Changing individual attributes does not affect any objects that were previously created using the attributes object.

RETURN VALUES

The created mutex attributes object is returned to the *attr* parameter.

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
: The value specified by *attr* is invalid.

`ENOMEM`
: Insufficient memory exists to create the mutex attributes object.

SEE ALSO

`pthread_create`(3-thr), `pthread_mutex_init`(3-thr), `pthread_mutexattr_delete`(3-thr), `pthread_mutexattr_getkind_np`(3-thr), `pthread_mutexattr_setkind_np`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_mutexattr_delete(3-thr)

NAME

`pthread_mutexattr_delete -` deletes a mutex attributes object

SYNOPSIS

```
#include <pthread.h>
int pthread_mutexattr_delete(pthread_mutexattr_t *attr);
```

PARAMETERS

attr Mutex attributes object deleted.

DESCRIPTION

The `pthread_mutexattr_delete()` routine deletes a mutex attributes object. Call this routine when a mutex attributes object is no longer referenced by the `pthread_mutexattr_create()` routine.

This routine gives permission to reclaim storage for the mutex attributes object. Mutexes that were created using this attributes object are not affected by the deletion of the mutex attributes object.

The results of calling this routine are unpredictable if the attributes object specified in the *attr* parameter does not exist.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
The value specified by *attr* is invalid.

SEE ALSO

`pthread_mutexattr_create`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_mutexattr_getkind_np(3-thr)

NAME

`pthread_mutexattr_getkind_np -` obtains the mutex type attribute usedwhen a mutex is created

SYNOPSIS

```
#include <pthread.h>
int pthread_mutexattr_getkind_np(pthread_mutexattr_t attr);
```

PARAMETERS

attr Mutex attributes object whose mutex type is obtained.

DESCRIPTION

The `pthread_mutexattr_getkind_np()` routine obtains the mutex type attribute that is used when a mutex is created. See the `pthread_mutexattr_setkind_np()` reference page for information about mutex type attributes.

This routine is a new primitive.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
: The value specified by *attr* is invalid.

SEE ALSO

`pthread_mutex_init`(3-thr) , `pthread_mutexattr_create`(3-thr) , `pthread_mutexattr_setkind_np`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_mutexattr_setkind_np(3-thr)

NAME

`pthread_mutexattr_setkind_np` - specifies the mutex type attribute

SYNOPSIS

```
#include <pthread.h>
int pthread_mutexattr_setkind_np(
pthread_mutexattr_t *attr,
int kind);
```

PARAMETERS

attr Mutex attributes object modified.

kind New value for the mutex type attribute. The *kind* parameter specifies the type of mutex that is created. Valid values are `MUTEX_FAST_NP` (default), `MUTEX_RECURSIVE_NP`, and `MUTEX_NONRECURSIVE_NP`.

DESCRIPTION

The `pthread_mutexattr_setkind_np()` routine sets the mutex type attribute that is used when a mutex is created.

A fast mutex is locked and unlocked in the fastest manner possible. A fast mutex can only be locked (obtained) once. All subsequent calls to `pthread_mutex_lock()` cause the calling thread to block until the mutex is freed by the thread that owns it. If the thread that owns the mutex attempts to lock it again, the thread waits for itself to release the mutex (causing a deadlock).

A recursive mutex can be locked more than once by the same thread without causing that thread to deadlock. In other words, a single thread can make consecutive calls to `pthread_mutex_lock()` without blocking. The thread must then call `pthread_mutex_unlock()` the same number of times as it called `pthread_mutex_lock()` before another thread can lock the mutex.

A nonrecursive mutex is locked only once by a thread, like a fast mutex. If the thread tries to lock the mutex again without first unlocking it, the thread receives an error. Thus, nonrecursive mutexes are more informative than fast mutexes because fast mutexes block in such a case, leaving it up to you to determine why the thread no longer executes. Also, if someone other than the owner tries to unlock a nonrecursive mutex, an error is returned.

Never use a recursive mutex with condition variables because the implicit unlock performed for a `pthread_cond_wait()` or `pthread_cond_timedwait()` might not actually release the mutex. In that case, no other thread can satisfy the condition of the predicate.

This routine is a new primitive.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
The value specified by *attr* is invalid.

`EPERM`
The caller does not have the appropriate privileges.

`ERANGE`
One or more parameters supplied have an invalid value.

SEE ALSO

`pthread_mutex_init`(3-thr) , `pthread_mutexattr_create`(3-thr) , `pthread_mutexattr_getkind_np`(3-thr) .

pthread_once(3-thr)

NAME

`pthread_once` - calls an initialization routine executed by onethread, a single time

SYNOPSIS

```
#include <pthread.h>
int pthread_once(□
pthread_once_t *once_block,□
pthread_initroutine_t init_routine);□
```

PARAMETERS

once_block

Address of a record that defines the one-time initialization code. Each one-time initialization routine must have its own unique `pthread_once_t` data structure.

init_routine

Address of a procedure that performs the initialization. This routine is called only once, regardless of the number of times it and its associated *once_block* are passed to `pthread_once()`.

DESCRIPTION

The `pthread_once()` routine calls an initialization routine executed by one thread, a single time. This routine allows you to create your own initialization code that is guaranteed to be run only once, even if called simultaneously by multiple threads or multiple times in the same thread.

For example, a mutex or a thread-specific data key must be created exactly once. Calling `pthread_once()` prevents the code that creates a mutex or thread-specific data from being called by multiple threads. Without this routine, the execution must be serialized so that only one thread performs the initialization. Other threads that reach the same point in the code are delayed until the first thread is finished.

This routine initializes the control record if it has not been initialized and then determines if the client one-time initialization routine has executed once. If it has not executed, this routine calls the initialization routine specified in *init_routine*. If the client one-time initialization code has executed once, this routine returns.

The `pthread_once_t` data structure is a record that allows client initialization operations to guarantee mutual exclusion of access to the initialization routine, and that each initialization routine is executed exactly once.

The client code must declare a variable of type `pthread_once_t` to use the client initialization operations. This variable must be initialized using the `pthread_once_init` macro, as follows:

```
static pthread_once_t myOnceBlock = pthread_once_init;
```

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`

The value specified by a parameter is invalid.

pthread_self(3-thr) - sigwait(3-thr)

pthread_self(3-thr)

NAME

`pthread_self` - obtains the identifier of the current thread

SYNOPSIS

```
#include <pthread.h>
pthread_t pthread_self();
```

DESCRIPTION

The `pthread_self()` routine allows a thread to obtain its own identifier. For example, this identifier allows a thread to set its own priority.

This value becomes meaningless when the thread object is deleted; that is, when the thread terminates its execution and `pthread_detach()` is called.

RETURN VALUES

Returns the identifier of the calling thread to `pthread_t`.

SEE ALSO

`pthread_create`(3-thr), `pthread_setprio`(3-thr), `pthread_setscheduler`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_setasynccancel(3-thr)

NAME

`pthread_setasynccancel -` enables or disables the current thread'sasynchronous cancelability

SYNOPSIS

```
#include <pthread.h>
int pthread_setasynccancel(int state);
```

PARAMETERS

state

State of asynchronous cancelability set for the calling thread. On return, receives the prior state of asynchronous cancelability. Valid values are as follows:

`CANCEL_ON`
: Asynchronous cancelability is enabled.

`CANCEL_OFF`
: Asynchronous cancelability is disabled.

DESCRIPTION

The `pthread_setasynccancel()` routine enables or disables the current thread's asynchronous cancelability and returns the previous asynchronous cancelability state.

When general cancelability is set to `CANCEL_OFF`, a cancel cannot be delivered to the thread, even if a cancelable routine is called or asynchronous cancelability is enabled. When general cancelability is set to `CANCEL_ON`, cancelability depends on the state of the thread's asynchronous cancelability.

When general cancelability is set to `CANCEL_ON` and asynchronous cancelability is set to `CANCEL_OFF`, the thread can only receive a cancel at specific cancellation points (for example, condition waits, thread joins, and calls to the `pthread_testcancel()` routine). If both general cancelability and asynchronous cancelability are set to `CANCEL_ON`, the thread can be canceled at any point in its execution.

When a thread is created, the default asynchronous cancelability state is `CANCEL_OFF`.

If you call this routine to enable asynchronous cancels, call it in a region of code where asynchronous delivery of cancels is disabled by a previous call to this routine. Do not call threads routines in regions of code where asynchronous delivery of cancels is enabled. The previous state of asynchronous delivery can be restored later by another call to this routine.

RETURN VALUES

On successful completion, the previous state of asynchronous cancelability is returned. If the function fails, −1 is returned. Following are the possible return values and the possible corresponding values (if any) for `errno`:

`CANCEL_ON`
: Asynchronous cancelability was on.

`CANCEL_OFF`
: Asynchronous cancelability was off.

`EINVAL`
: The specified state is not `CANCEL_ON` or `CANCEL_OFF`.

SEE ALSO

`pthread_cancel`(3-thr), `pthread_setcancel`(3-thr), `pthread_testcancel`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_setcancel(3-thr)

NAME

`pthread_setcancel` - enables or disables the current thread's generalcancelability

SYNOPSIS

```
#include <pthread.h>
int pthread_setcancel(int state);
```

PARAMETERS

state

State of general cancelability set for the calling thread. On return, receives the prior state of general cancelability. Valid values are as follows:

`CANCEL_ON`
General cancelability is enabled.

`CANCEL_OFF`
General cancelability is disabled.

DESCRIPTION

The `pthread_setcancel()` routine enables or disables the current thread's general cancelability and returns the previous general cancelability state.

When general cancelability is set to `CANCEL_OFF`, a cancel cannot be delivered to the thread, even if a cancelable routine is called or asynchronous cancelability is enabled.

When a thread is created, the default general cancelability state is `CANCEL_ON`.

Possible Dangers of Disabling Cancelability

The most important use of cancels is to ensure that indefinite wait operations are terminated. For example, a thread waiting on some network connection, which may take days to respond (or may never respond), is normally made cancelable.

However, when cancelability is disabled, no routine is cancelable. Waits must be completed normally before a cancel can be delivered. As a result, the program stops working and the user is unable to cancel the operation.

When disabling cancelability, be sure that no long waits can occur or that it is necessary for other reasons to defer cancels around that particular region of code.

RETURN VALUES

On successful completion, the previous state of general cancelability is returned. If the function fails, −1 is returned. Following are the possible return values and the possible corresponding values (if any) for `errno`:

`CANCEL_ON`
General cancelability was on.

`CANCEL_OFF`
General cancelability was off.

`EINVAL`
The specified state is not `CANCEL_ON` or `CANCEL_OFF`.

SEE ALSO

`pthread_cancel`(3-thr), `pthread_setasynccancel`(3-thr), `pthread_testcancel`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_setprio(3-thr)

NAME

`pthread_setprio -` changes the current priority of a thread

SYNOPSIS

```
#include <pthread.h>
int pthread_setprio(
pthread_t thread,
int priority);
```

PARAMETERS

thread

Thread whose priority is changed.

priority

New priority value of the thread specified in *thread*. The priority value depends on scheduling policy. Valid values fall within one of the following ranges:

`PRI_OTHER_MIN` **<=** *priority* **<=** `PRI_OTHER_MAX`

`PRI_FIFO_MIN` **<=** *priority* **<=** `PRI_FIFO_MAX`

`PRI_RR_MIN` **<=** *priority* **<=** `PRI_RR_MAX`

`PRI_FG_MIN_NP` **<=** *priority* **<=** `PRI_FG_MAX_NP`

`PRI_BG_MIN_NP` **<=** *priority* **<=** `PRI_BG_MAX_NP`

If you create a new thread without specifying a threads attributes object that contains a changed priority attribute, the default priority of the newly created thread is the midpoint between `PRI_OTHER_MIN` and `PRI_OTHER_MAX` (the midpoint between the minimum and the maximum for the `SCHED_OTHER` policy).

When you call this routine to specify a minimum or maximum priority, use the appropriate symbol; for example, `PRI_FIFO_MIN` or `PRI_FIFO_MAX`. To specify a value between the minimum and maximum, use an appropriate arithmetic expression. For example, to specify a priority midway between the minimum and maximum for the Round Robin scheduling policy, specify the following concept using your programming language's syntax:

```
pri_rr_mid = (PRI_RR_MIN + PRI_RR_MAX + 1)/2
```

If your expression results in a value outside the range of minimum to maximum, an error results when you use it.

DESCRIPTION

The `pthread_setprio()` routine changes the current priority of a thread. A thread can change its own priority using the identifier returned by `pthread_self()`.

Changing the priority of a thread can cause it to start executing or be preempted by another thread. The effect of setting different priority values depends on the scheduling priority assigned to the thread. The initial scheduling priority is set by calling the `pthread_attr_setprio()` routine.

Note that `pthread_attr_setprio()` sets the priority attribute that is used to establish the priority of a new thread when it is created. However, `pthread_setprio()` changes the priority of an existing thread.

RETURN VALUES

If successful, this routine returns the previous priority. If the function fails, `errno` may be set to one of the following values:

`EINVAL`

The value specified by *thread* is invalid.

`ENOTSUP`

An attempt is made to set the priority to an unsupported value.

`ESRCH`
The value specified by *thread* does not refer to an existing thread.

`EPERM`
The caller does not have the appropriate privileges to set the priority of the specified thread.

SEE ALSO

`pthread_attr_setprio`(3-thr), `pthread_attr_setsched`(3-thr), `pthread_create`(3-thr), `pthread_self`(3-thr), `pthread_setscheduler`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_setscheduler(3-thr)

NAME

pthread_setscheduler - changes the current scheduling policy andpriority of a thread

SYNOPSIS

```
#include <pthread.h>
int pthread_setscheduler(
pthread_t thread,
int scheduler,
int priority);
```

PARAMETERS

thread

Thread whose scheduling policy is to be changed.

scheduler

New scheduling policy value for the thread specified in *thread*. Valid values are as follows:

SCHED_FIFO

(First In, First Out) The highest-priority thread runs until it blocks. If there is more than one thread with the same priority, and that priority is the highest among other threads, the first thread to begin running continues until it blocks.

SCHED_RR

(Round Robin) The highest-priority thread runs until it blocks; however, threads of equal priority, if that priority is the highest among other threads, are timesliced. Timeslicing is a process in which threads alternate using available processors.

SCHED_OTHER

(Default) All threads are timesliced. SCHED_OTHER ensures that all threads, regardless of priority, receive some scheduling, and thus no thread is completely denied execution time. (However, SCHED_OTHER threads can be denied execution time by SCHED_FIFO or SCHED_RR threads.)

SCHED_FG_NP

(Foreground) Same as SCHED_OTHER. Threads are timesliced and priorities can be modified dynamically by the scheduler to ensure fairness.

SCHED_BG_NP

(Background) Like SCHED_OTHER, ensures that all threads, regardless of priority, receive some scheduling. However, SCHED_BG_NP can be denied execution by any of the other scheduling policies.

priority

New priority value of the thread specified in *thread*. The priority attribute depends on scheduling policy. Valid values fall within one of the following ranges:

PRI_OTHER_MIN <= *priority* <= PRI_OTHER_MAX

PRI_FIFO_MIN <= *priority* <= PRI_FIFO_MAX

PRI_RR_MIN <= *priority* <= PRI_RR_MAX

PRI_FG_MIN_NP <= *priority* <= PRI_FG_MAX_NP

PRI_BG_MIN_NP <= *priority* <= PRI_BG_MAX_NP

If you create a new thread without specifying a threads attributes object that contains a changed priority attribute, the default priority of the newly created thread is the midpoint between PRI_OTHER_MIN and PRI_OTHER_MAX (the midpoint between the minimum and the maximum for the SCHED_OTHER policy).

When you call this routine to specify a minimum or maximum priority, use the appropriate symbol; for

example, PRI_FIFO_MIN or PRI_FIFO_MAX. To specify a value between the minimum and maximum, use an appropriate arithmetic expression. For example, to specify a priority midway between the minimum and maximum for the Round Robin scheduling policy, specify the following concept using your programming language's syntax:

```
pri_rr_mid = (PRI_RR_MIN + PRI_RR_MAX)/2
```

If your expression results in a value outside the range of minimum to maximum, an error results when you use it.

DESCRIPTION

The pthread_setscheduler() routine changes the current scheduling policy and priority of a thread. Call this routine to change both the priority and scheduling policy of a thread at the same time. To change only the priority, call the pthread_setprio() routine.

A thread changes its own scheduling policy and priority by using the identifier returned by pthread_self(). Changing the scheduling policy or priority, or both, of a thread can cause it to start executing or to be preempted by another thread.

This routine differs from pthread_attr_setprio() and pthread_attr_setsched() because those routines set the priority and scheduling policy attributes that are used to establish the priority and scheduling policy of a new thread when it is created. This routine, however, changes the priority and scheduling policy of an existing thread.

RETURN VALUES

If successful, the previous scheduling policy value is returned. If the function fails, errno may be set to one of the following values:

EINVAL
: The value specified by *thread* is invalid.

ENOTSUP
: An attempt is made to set the policy to an unsupported value.

ESRCH
: The value specified by *thread* does not refer to an existing thread.

EPERM
: The caller does not have the appropriate privileges to set the scheduling policy of the specified thread.

SEE ALSO

pthread_attr_setprio(3-thr), pthread_attr_setsched(3-thr), pthread_create(3-thr), pthread_self(3-thr), pthread_setprio(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_setspecific(3-thr)

NAME

`pthread_setspecific -` sets the thread-specific data associated withthe specified key for the current thread

SYNOPSIS

```
#include <pthread.h>
int pthread_setspecific(
pthread_key_t key,
pthread_addr_t value);
```

PARAMETERS

key Context key value that uniquely identifies the context value specified in *value*. This key value must have been obtained from `pthread_keycreate()`.

value
Address containing data to be associated with the specified key for the current thread; this is the thread-specific data.

DESCRIPTION

The `pthread_setspecific()` routine sets the thread-specific data associated with the specified key for the current thread. If a value has already been defined for the key in this thread, the new value is substituted for it.

Different threads can bind different values to the same key. These values are typically pointers to blocks of dynamically allocated memory that are reserved for use by the calling thread.

RETURN VALUES

If the function fails, –1 is returned, and `errno` may be set to the following value:

`EINVAL`
The key value is invalid.

SEE ALSO

`pthread_getspecific`(3-thr) , `pthread_keycreate`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_signal_to_cancel_np(3-thr)

NAME

`pthread_signal_to_cancel_np -` cancels the specified thread

SYNOPSIS

```
#include <pthread.h>
int pthread_signal_to_cancel_np(
sigset_t *sigset,
pthread_t *thread);
```

PARAMETERS

sigset
: Signal mask containing a list of signals that, when received by the process, cancels the specified thread.

thread
: The thread canceled if a valid signal is received by the process.

DESCRIPTION

The `pthread_signal_to_cancel_np()` routine requests that the specified thread be canceled if one of the signals specified in the signal mask is received by the process. The set of legal signals is the same as that for the `sigwait()` service. The *sigset* parameter is not validated. If it is invalid, this routine returns successfully but neither the specified thread nor the previously specified thread is canceled if a signal occurs.

Note that the address of the specified thread is saved in a per-process global variable. Therefore, any subsequent call to this routine by your application or any library function will supersede the thread specified in the previous call, and that thread will not be canceled if one of the signals specified for it is delivered to the process. In other words, take care when you call this routine; if another thread calls it after you do, the expected result of this routine will not occur.

RETURN VALUES

If the function fails, `errno` may be set to one of the following values:

`EINVAL`
: The value specified by *thread* is invalid.

SEE ALSO

`pthread_cancel`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

pthread_testcancel(3-thr)

NAME

`pthread_testcancel -` requests delivery of a pending cancel to thecurrent thread

SYNOPSIS

```
#include <pthread.h>
void pthread_testcancel();
```

DESCRIPTION

The `pthread_testcancel()` routine requests delivery of a pending cancel to the current thread. The cancel is delivered only if a cancel is pending for the current thread and general cancel delivery is not currently disabled. (A thread disables delivery of cancels to itself by calling the `pthread_setcancel()` routine.)

This routine, when called within very long loops, ensures that a pending cancel is noticed within a reasonable amount of time.

RETURN VALUES

No value is returned.

SEE ALSO

`pthread_cancel`(3-thr), `pthread_setasynccancel`(3-thr), `pthread_setcancel`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_unlock_global_np(3-thr)

NAME

`pthread_unlock_global_np -` unlocks a global mutex

SYNOPSIS

```
#include <pthread.h>
void pthread_unlock_global_np();
```

DESCRIPTION

The `pthread_unlock_global_np()` routine unlocks the global mutex when each call to `pthread_lock_global_np()` is matched by a call to this routine. For example, if you called `pthread_lock_global_np()` three times, `pthread_unlock_global_np()` unlocks the global mutex when you call it the third time. If no threads are waiting for the global mutex, it becomes unlocked with no current owner. If one or more threads are waiting to lock the global mutex, one thread returns from its call to `pthread_lock_global_np()`. The scheduling policy is used to determine which thread acquires the global mutex. For the policies `SCHED_FIFO` and `SCHED_RR`, a blocked thread is chosen in priority order.

The results of calling this routine are unpredictable if the global mutex is already unlocked. The results of calling this routine are also unpredictable if the global mutex is owned by a thread other than the calling thread.

This routine is a new primitive.

RETURN VALUES

No value is returned.

SEE ALSO

`pthread_lock_global_np`(3-thr), `pthread_mutex_lock`(3-thr), `pthread_mutex_unlock`(3-thr), `pthread_mutexattr_setkind_np`(3-thr).

Fujitsu Siemens Computers Add-on Reference Manual

pthread_yield(3-thr)

NAME

`pthread_yield -` notifies the scheduler that the current thread iswilling to release its processor

SYNOPSIS

```
#include <pthread.h>
void pthread_yield();
```

DESCRIPTION

The `pthread_yield()` routine notifies the scheduler that the current thread is willing to release its processor to other threads of the same priority. (A thread releases its processor to a thread of a higher priority without calling this routine.)

If the current thread's scheduling policy (as specified in a call to the `pthread_attr_setsched()` or `pthread_setscheduler()` routine) is `SCHED_FIFO` or `SCHED_RR`, this routine yields the processor to other threads of the same or a higher priority. If no threads of the same priority are ready to execute, the thread continues.

This routine allows knowledge of the details of an application to be used to increase fairness. It increases fairness of access to the processor by removing the current thread from the processor. It also increases fairness of access to shared resources by removing the current thread from the processor as soon as it is finished with the resource.

Call this routine when a thread is executing code that denies access to other threads on a uniprocessor if the scheduling policy is `SCHED_FIFO`.

Use `pthread_yield()` carefully because misuse causes unnecessary context switching, which increases overhead without increasing fairness. For example, it is counterproductive for a thread to yield while it has a needed resource locked.

RETURN VALUES

No value is returned.

SEE ALSO

`pthread_attr_setsched`(3-thr) , `pthread_setscheduler`(3-thr) .

Fujitsu Siemens Computers Add-on Reference Manual

sigaction(3-thr)

NAME

`sigaction -` examines and changes synchronous signal actions (POSIXsoftware signal facilities)

SYNOPSIS

```
#include <signal.h>
struct sigaction {
    void (*sa_handler)();
    sigset_t sa_mask;
    int sa_flags;
};

int sigaction(int sig; const struct sigaction *act;
                struct sigaction *oact);
```

PARAMETERS

sig Synchronous signal to examine or change.

act Points to a `sigaction` structure that describes the action to be taken upon receipt of the signal indicated by the value of the *act* parameter.

oact Points to a `sigaction` structure in which the signal action data in effect at the time of the `sigaction()` function call is returned.

DESCRIPTION

The `sigaction` POSIX service allows for per-thread handlers to be installed for catching synchronous signals. It is called in a multithreaded process to establish thread specific actions for such signals. This call is the POSIX equivalent of the `sigaction()` system call with the following exceptions or modifications:

- The `sigaction()` routine only modifies behavior for individual threads.
- The `sigaction()` routine only works for synchronous signals. Attempting to set a signal action for an asynchronous signal is an error. This is true even in a single-threaded process.

  Any multithreaded application using DCE Threads will need to use the `sigwait()` function for dealing with asynchronous signals. The `sigwait()` function can be used to synchronously wait for delivery of asynchronously generated signals.
- The `SA_RESTART` flag is always set by the underlying system in POSIX mode so that interrupted system calls will fail with return value of –1 and the `EINTR` error in `errno` instead of getting restarted.

  The system's `SA_RESTART` flag has the opposite meaning of the `SA_RESTART` flag in the *sa_flags* field and is always set in the underlying system call resulting from `sigaction()` regardless of whether `SA_RESTART` was indicated in *sa_flags*.
- The signal mask is manipulated using the POSIX `sigsetops()` functions. They are `sigemptyset()`, `sigfillset()`, `sigaddset()`, `sigdelset()`, and `sigismember()`.

The `sigaction()` function can be used to inquire about the current handling of a given signal by specifying a null pointer for *act*, since the action is unchanged unless this parameter is not a null pointer. In order for the signal action in effect at the time of the `sigaction()` call to be returned, the *oact* parameter must not be a null pointer.

RETURN VALUES

Possible return values are as follows:

`EFAULT`
: Either *act* or *oact* points to memory which is not a valid part of the process address space. A new signal

handler is not installed.

`EINVAL`
The value specified by *sig* is invalid. A new signal handler is not installed.

`EINVAL`
An attempt is made to ignore or supply a handler for `SIGKILL` or `SIGSTOP`. A new signal handler is not installed.

SEE ALSO

`sigsuspend`(2), `sigvec`(2), `siginterrupt`(3), `sigpending`(3-thr), `sigprocmask`(3-thr), `setjmp`(3C), `sigsetops`(3C), `signal`(5), `tty`(7).

Fujitsu Siemens Computers Add-on Reference Manual

sigpending(3-thr)

NAME

`sigpending` - examines pending signals (POSIX software signalfacilities)

SYNOPSIS

```
#include <signal.h>
int sigpending(sigset_t *set);
```

PARAMETERS

set Points to a location in which the signals that are blocked from delivery and pending at the time of the `sigpending()` function call are returned.

DESCRIPTION

The `sigpending()` function stores the set of signals that are blocked from delivery and pending for the calling process in the space pointed to by the argument *set*.

The `sigpending()` function may be called by any thread in a multithreaded process to determine which signals are in the pending set for that thread. Since DCE Threads supports the `{_POSIX_THREADS_PER_PROCESS_SIGNALS_1}` option, signals pending upon the thread are those that are pending upon the process.

RETURN VALUES

Possible return values are as follows:

`EFAULT`
The *set* argument points to memory that is not a valid part of the process address space.

SEE ALSO

`sigprocmask`(3-thr), `sigsetops`(3C), `signal`(5).

Fujitsu Siemens Computers Add-on Reference Manual

sigprocmask(3-thr)

NAME

`sigprocmask` - examines and changes blocked signals (POSIX softwaresignal facilities)

SYNOPSIS

```
#include <signal.h>
int sigprocmask(int how, const sigset_t *set, sigset_t
*oset);
```

PARAMETERS

how The manner in which the values in *set* are changed as defined by one of the described argument values.

set A set of signals that will be used to change the current thread's signal mask according to the value in the *how* parameter.

oset Points to a location in which the signal mask in effect at the time of the `sigprocmask()` function call is returned.

DESCRIPTION

The `sigprocmask()` function is used to examine or change (or both) the signal mask of the calling process. If the value of the argument *set* is not `NULL`, it points to a set of signals to be used to change the currently blocked set according to the *how* parameter as follows:

`SIG_BLOCK`
The resulting signal set is the union of the current set and the signal set pointed to by the argument *set*.

`SIG_UNBLOCK`
The resulting signal set is the intersection of the current set and the and the complement of the signal set pointed to by the argument *set*.

`SIG_SETMASK`
The resulting signal set is the signal set pointed to by the argument *set*.

If the argument *oset* is not `NULL`, the previous mask is stored in the space pointed to by *oset*.

The `sigprocmask()` function can be used to inquire about the currently blocked signals by specifying a null pointer for *set*, since the value of the argument *how* is not significant and the signal mask of the process is unchanged unless this parameter is not a null pointer. In order for the signal mask in effect at the time of the `sigprocmask()` call to be returned, the *oset* argument must not be a null pointer.

If there are any pending unblocked signals after the call to the `sigprocmask()` function, at least one of those signals shall be delivered before the `sigprocmask()` function returns. As a system restriction, the `SIGKILL` and `SIGSTOP` signals cannot be blocked.

If the `sigprocmask()` function fails, the signal mask of the process is not changed by this function call.

RETURN VALUES

Possible return values are as follows:

`EINVAL`
The value specified by the *how* parameter is not equal to one of the defined values. The signal mask of the process remains unchanged.

SEE ALSO

`sigsuspend`(2), `sigaction`(3-thr), `sigpending`(3-thr), `sigsetops`(3C), `signal`(5).

sigwait(3-thr)

NAME

`sigwait -` causes a thread to wait for an asynchronous signal

SYNOPSIS

```
#include <pthread.h>
int sigwait(sigset_t *set);
```

PARAMETERS

set Set of pending signals upon which the calling thread will wait.

DESCRIPTION

This routine causes a thread to wait for an asynchronous signal. It atomically chooses a pending signal from *set*, atomically clears it from the system's set of pending signals and returns that signal number. If no signal in *set* is pending at the time of the call, the thread is blocked until one or more signals becomes pending. The signals defined by *set* may be unblocked during the call to this routine and will be blocked when the thread returns from the call unless some other thread is currently waiting for one of those signals.

A thread must block the signals it waits for using `sigprocmask()` prior to calling this function.

If more than one thread is using this routine to wait for the same signal, only one of these threads will return from this routine with the signal number.

A call to `sigwait()` is a cancellation point.

RETURN VALUES

Possible return values are as follows:

`EINVAL`
: One or more of the values specified by *set* is invalid.

`EINVAL`
: One or more of the values specified by *set* is not blocked.

`EINVAL`
: There are no values specified in *set*.

SEE ALSO

`pause(2)`, `sigpending(2)`, `sigprocmask(2)`, `pthread_cancel(3-thr)`, `pthread_setasynccancel(3-thr)`, `sigsetops(3C)`, `signal(5)`.

File Formats(4)

cluster(4)

NAME

`cluster -` configuration file

SYNOPSIS

`/etc/default/cluster`

DESCRIPTION

The `/etc/default/cluster` file contains configuration information which is analyzed by the Cluster Configuration Daemon `CCfgD`. It contains one or more configuration blocks, where each block contains the

cluster node parameters for a particular system. Each block is given a name. Only the configuration block with the current system name (as given by `uname –n`) or with the special name `@node` is relevant for a particular system.

General format of the file

White space:

Lines containing only white space are ignored. Keywords and arguments are separated by one or more white space characters.

Comments:

Lines starting with a # character are comment lines; they are ignored. Lines within a configuration block may also contain comments that are introduced by a white space character followed by a # character. All characters beginning with # up to the end of the line are ignored.

Configuration blocks:

name { *config-line ...* }

Each configuration block is introduced by a name, followed by a number of configuration entries enclosed in braces. The entries within a block are line oriented.

The name of the configuration block is used to find the configuration block for a particular system. There are two possibilities:

- The file contains a number of configuration blocks. Each one is labeled with a system name. The configuration block for the local system is found by matching the block name with the local system name (as given by `uname –n`).
- The file only contains a configuration block for the local system. In this case, the special name `@node` may also be used.

When the first format is used, it is possible to have one cluster-wide configuration file containing the blocks for all nodes of a cluster. The file may be edited on one node and distributed to all other nodes in a cluster.

Configuration entries within a block

A configuration block must contain the entire information needed by the `CCfgD` program to bootstrap a system as a cluster node. The block entries are line oriented. They are given as a keyword followed by one or more arguments.

The order of the entries is not important. Some entries are mandatory, others are optional and will be assigned default values when not specified.

`nodenum` *node-number*

Mandatory. The unique cluster node number.

`clustername` *cluster-name*

Optional. The cluster name. If not specified it defaults to `cluster`.

`icf` *ctlr net-type*:*device*[:*arguments*]

One entry is mandatory. Up to 4 entries are possible for 100 MBit Ethernet and FDDI Cluster Interconnect and 2 for an SCI Cluster Interconnect.

Description of a logical ICF device and the mapping to a physical network device:

Up to 4 (for SCI Cluster Interconnect 2) logical ICF devices may be defined. The first *ctlr* argument is the controller number of the logical ICF device. It must be in the range 0 ... 3.

The second argument is a description of the physical device to which the logical ICF device is mapped. It consists of two or more parameters separated by the : character (colon).

net-type

Type of the network device.

The ICF devices are mapped to DLPI network interfaces for 100 MBit Ethernet or FDDI Cluster Interconnect. The keyword `DLPI` is used.

The ICF devices are mapped to SCI controller numbers for SCI Cluster Interconnect. The keyword `SCI` is used.

device

Name of the network device.

For 100 MBit Ethernet or FDDI Cluster Interconnect, this is the device node name of the DLPI interface, for example `/dev/dlpi/zx1`.

For SCI Cluster Interconnect, this is the SCI controller number. A system usually has 2 SCI controllers, 0 and 1.

arguments

Optional. Any additional arguments that may be needed to set up the ICF-to-physical-device mapping.

No additional arguments are required for the interfaces currently supported.

`ip` *ip-address* [`netmask` *ip-netmask*] [`broadcast` *ip-bcast-addr*]

Mandatory. The IP address to be used to initialize the IP-ICF interface. This may be a name or an IP address in decimal point notation. When the interface address is given as a name, that name must resolve into an IP address using `gethostbyname(3N)` , e.g. it must exist in the `/etc/hosts` file or/and in the DNS database.

Optionally a netmask and a broadcast address may be specified for the interface.

The *ip-netmask* may be given either as a hexadecimal number or in decimal point notation.

The *ip-bcast-addr* must be given in dotted quad notation.

PHYSICAL CONNECTION TO THE CLUSTER INTERCONNECT

The definition of logical ICF devices in the cluster configuration file specifies the rules for physically connecting SCI or network controllers.

All SCI (or network) controllers that map to the same logical ICF controller number must connect to the same physical SCI ring or physical network.

EXAMPLES

```
## Minimum cluster configuration file
#
# Two logical ICF devices are defined:
#     ICF 0 mapped to the DLPI interface /dev/dlpi/zx1
#     ICF 1 mapped to the DLPI interface /dev/dlpi/zx2
#
# The cluster name defaults to "cluster"
#

@node
{
 nodenum 3
 icf 0 DLPI:/dev/dlpi/zx1
 icf 1 DLPI:/dev/dlpi/zx2
 ip 192.168.0.3
}

# ------------------------------------------------------------------------

#
# More complex cluster configuration file
#
# This would be usable on the three hosts
# alpha, beta, and gamma
#
```

```
#   alpha:  node number 1, IP address alpha-ic
#   beta:   node number 2, IP address beta-ic
#   gamma:  node number 4, IP address gamma-ic
#
# The cluster name is "testcluster"
#

alpha
{
 nodenum 1
 clustername  testcluster
 icf 0 DLPI:/dev/dlpi/zx1
 icf 1 DLPI:/dev/dlpi/zx2
 ip cl-1-ic netmask 0xFFFFFF80
}

beta
{
 nodenum 2
 clustername  testcluster
 icf 0 DLPI:/dev/dlpi/zx3
 icf 1 DLPI:/dev/dlpi/zx4
 ip cl-2-ic netmask 0xFFFFFF80
}

gamma
{
 nodenum 4
 clustername  testcluster
 icf 0 DLPI:/dev/dlpi/zx1
 icf 1 DLPI:/dev/dlpi/zx2
 ip cl-4-ic netmask 0xFFFFFF80
}

# ------------------------------------------------------------------------

#
# Cluster configuration file for SCI as Cluster Interconnect
#
# This would be usable on the two hosts
# alpha and beta
#
#   alpha:  node number 1, IP address alpha-ic
#   beta:   node number 2, IP address beta-ic
#
# The cluster name is "testcluster"
#

alpha
{
 nodenum 1
 clustername  testcluster
 icf 0 SCI:0
 icf 1 SCI:1
 ip cl-1-ic netmask 0xFFFFFF80
}

beta
{
```

```
 nodenum 2
 clustername  testcluster
 icf 0 SCI:0
 icf 1 SCI:1
 ip cl-2-ic netmask 0xFFFFFF80
}

#
# Note on connecting the systems to physical networks or SCI rings
#
#   The cluster connection for the three systems alpha, beta, and gamma
#   consists of 2 100 Mbit/s LANs, LAN0 and LAN1.
#   All physical controllers on the three machines that map to
#   the logical ICF device 0 must be connected to LAN0, all controllers
#   that map to the logical ICF device 1 must be connected to LAN1.
#
#   alpha:  icf 0 <-> zx1 ---> LAN0
#           icf 1 <-> zx2 ----------> LAN1
#
#   beta:   icf 0 <-> zx3 ---> LAN0
#           icf 1 <-> zx4 ----------> LAN1
#
#   gamma:  icf 0 <-> zx1 ---> LAN0
#           icf 1 <-> zx2 ----------> LAN1
#
#
#   The same is true when using SCI as Cluster Interconnect.
#   Assume a cluster consisting of two nodes, alpha and beta,
#   each having 2 SCI adapters.
#
#   alpha:  icf 0 <-> SCI controller-0 ---> SCI ring 0
#           icf 1 <-> SCI controller-1 --------------> SCI ring 1
#
#   beta:   icf 0 <-> SCI controller-0 ---> SCI ring 0
#           icf 1 <-> SCI controller-1 --------------> SCI ring 1
#

# ----------------------------------------------------------------------
```

FILES

/etc/default/cluster

SEE ALSO

uname(1), CCfgD(1M).

Fujitsu Siemens Computers Add-on Reference Manual

clustertab(4)
SIDLM

NAME

`clustertab` - cluster configuration file for the Distributed LockManager

DESCRIPTION

The `/etc/clustertab` file describes the configuration of a cluster in the SIDLM cluster, and it is used by administrative tools, such as `dlmconfig(8)` , to administer and configure the Distributed Lock Manager (DLM) cluster environment.

The system administrator is responsible for maintaining this file correctly and consistently across all nodes within the cluster. This file should be well protected, and a backup copy should be made after each modification.

The system administrator can modify this file with a text editor. Within this file, fields are separated by white space. A pound sign (#) in the first column of a line means that line is a comment. Blank lines are ignored. The order of the node definitions is insignificant; however, having a similar file arrangement across all nodes makes administration easier.

All device names used in `/etc/clustertab` must already exist and be accessible by the `dlmconfig(8)` user. It is not necessary to use the same name for the device that supports the private network of the cluster; however, using the same device names makes administration easier.

CLUSTER CONFIGURATION

The `/etc/clustertab` configuration file must have one line for each node in the cluster. Each line specifies a unique node identifier, the node name, the device name, the controller number within the device, and the network address.

Each node identifier must be unique within each cluster. If the node identifiers are not unique, the cluster may perform as if it were two separate clusters/nodes. Since the current maximum cluster size is four systems, the node identifier for each system should be one of the following: 1, 2, 3, or 4. It is not necessary to assign consecutive node identifiers.

The node name is used to symbolically identify the node and is also used to communicate with the system through the Local Area Network (LAN).

The device field is the device name of the local communications device through which private messages are sent to the destination node. If the MAC switch driver is used, the device name is `/dev/msw/swether`. Otherwise, this is `/dev/mc2/mc2ether`. The hardware on each system does not need to be arranged in the same way as the software (virtual disks, applications on single drives, etc.), but it is highly recommended for ease of administration.

The controller number further specifies the port to be used within the device. If the MAC switch driver is used, this is the number of the MAC switch driver specified in `mswtab(4)` (e.g. 0 for `sw0`). Otherwise, this is the number of the Ethernet controller port of a physical controller as shown in the field "log" of the output of the command `showconf(8)` (e.g. 3 for `lce3`).

The network address is used when the local node needs to communicate with the remote node in the cluster over the private network. The local node sends the messages to the network address of the remote node. To obtain the network address, run `showconf`.

EXAMPLES

This example shows an `/etc/clustertab` file for a 2-node cluster without `msw`:

```
NodeID       NodeName        Device  Cntler  Network Address□
#1      system1 /dev/mc2/mc2ether       1       8:0:6:5:51:01□
#2      system2 /dev/mc2/mc2ether       1       8:0:6:5:51:02□
1       harry   /dev/mc2/mc2ether       1       8:0:6:5:51:01□
```

```
2       sally   /dev/mc2/mc2ether       1       8:0:6:5:51:02
```

The following example shows an `/etc/clustertab` file for a 4-node cluster with `msw` configured:

```
NodeID       NodeName        Device  Cntler  Network Address
1       harry   /dev/msw/swether        0       8:0:6:5:51:01
2       sally   /dev/msw/swether        0       8:0:6:5:51:02
3       mork    /dev/msw/swether        0       8:0:6:5:51:03
4       mindy   /dev/msw/swether        0       8:0:6:5:51:04
```

FILES

`/etc/clustertab`
cluster configuration file

SEE ALSO

`etherstat`(1M), `mc2tab`(4), `mswtab`(4), `dlm`(7), `dlmconfig`(8), `mswconfig`(8), `showconf`(8).

Fujitsu Siemens Computers Add-on Reference Manual

clustertab(4)
SIcluster

NAME

`clustertab` - cluster configuration file for OLR

SYNOPSIS

`/etc/clustertab`

DESCRIPTION

The `/etc/clustertab` file describes the configuration of a cluster and is used by OLR.

The system administrator is responsible for maintaining this file correctly and consistently across all nodes within the cluster. This file should be well protected, and a backup copy should be made after each modification.

The system administrator can modify this file with a text editor. Within this file, fields are separated by white space. A pound sign (#) in the first column of a line means that line is a comment. Blank lines are ignored. The order of the node definitions is insignificant; however, having a similar file arrangement across all nodes makes administration easier.

CLUSTER CONFIGURATION

The `/etc/clustertab` configuration file must have one line for each node in the cluster. Each line specifies a unique node identifier and a unique node name.

Node number and node identifier must be the same as in `/etc/default/cluster`.

EXAMPLES

This example shows an `/etc/clustertab` file for a 2-node cluster:

```
NodeID          NodeName
#1              system1
#2              system2
1               harry
2               sally
```

The following example shows an `/etc/clustertab` file for a 4-node cluster:

```
NodeID          NodeName
1               harry
2               sally
3               mork
4               mindy
```

FILES

`/etc/clustertab`
cluster configuration file

SEE ALSO

`cluster(4)` .

Fujitsu Siemens Computers Add-on Reference Manual

COD_prototype(4)
Cluster Operating System

NAME

COD_prototype - configuration file for the cluster object database

DESCRIPTION

The cluster operating system, CLOS, [see clos(7)] requires data to be available on all cluster nodes from which it takes its administration information. This data is stored in one or more files, which are collectively referred to as a cluster object database [CLODB, see clodb(7)]. The most important task for the CLODB subsystem is to distribute and maintain the consistency of these files. An application can specify in a COD_prototype file, which files must be placed for it in CLODB. These COD_prototype files are evaluated by CLOS when a cluster is created. Likewise, when re-installing a package, which contains a COD_prototype, an evaluation should be performed subsequently using the cod_prototype(1M) command. Should it happen that the command is not installed or the cluster is not set up, no evaluation need take place.

A COD_prototype file must be stored in the /opt/COD/prototypes directory for the evaluation. The name can be chosen freely, whereby the name of the package that installed it is recommended. No evaluation need be invoked in the case of systems on which the prototype directory is not available. Nevertheless, the directory should be created and the COD_prototype file installed.

STRUCTURE

Each COD_prototype file is evaluated line-by-line up to the "#" character or the line end "NL". The # character introduces a comment up to the line end.

The COD_prototype entry has the following structure per line:

CLOS_name : *flags* : *file* : *args*

The ":" (colon) character is used as a separator between the fields. The leading blank is ignored in each field.

Field descriptors can be used in the fields of an entry. These descriptors have the form %*character*. They are replaced by their value during the evaluation. Permitted *character* descriptors include the following:

A The argument of the –u option is used. For the remaining options, it is the same as H.

C For the respective CLOS name. If this name appears, the evaluation program uses the name of the cluster from the program parameter.

H For the respective system name (*nodename*) [see uname(1)]. If this name appears, the evaluation program replaces this descriptor with the value of *nodename*.

% The % character itself is used.

The individual fields of an entry have the following meaning:

CLOS_name

The name of the cluster for which this line is to be evaluated is entered in this field. If the cluster is not available on the node, this type of entry is ignored by the evaluation. If the field is blank, the entry is evaluated for each cluster.

flags

This field controls which actions must be executed for the evaluation. One or more identifiers are permitted in this field. These are as follows:

a When the cod_prototype(1M) evaluation program is called with the –a option, it executes the file in the *file* field as a command with the arguments in the *args* field. cod_prototype –a is always invoked when a new node is added to the cluster with node_add(1M) .

c During configuration, the cod_prototype(1M) evaluation program copies the file in the *file*

field to the CLODB of the identified cluster with the same pathname, if this is not yet configured. The absolute pathname of the file in the *file* field is appended to the /var/clos/*clos_name*/root home directory of CLODB. The permissions of the newly created file are taken from the *args* field of the COD_prototype file entry.

When cod_prototype(1M) is called with the –r (remove) option, the file is deleted in CLODB.

i The evaluation program executes the file in the *file* field when configuring (installing) CLODB as a command with the arguments in the *args* field.

Commands that can be called for the purpose of configuration must be of such a type that they can be called repeatedly at different times. The reason for this is that new applications are installed subsequently which install their COD_prototype files and then invoke the cod_prototype(1M) evaluation command for configuration.

l During configuration (installation), the evaluation program creates a symbolic link between the file in the *file* field and the CLODB of the identified cluster with the same pathname, if this is not already configured. The absolute pathname of the file in *file* field is appended to the /var/clos/*clos_name*/root home directory of CLODB. If cod_prototype(1M) is invoked with the –r (remove) option, the file is deleted in CLODB.

r The evaluation program executes the file in the *file* field when removing CLODB as a command with the arguments in the *args* field.

p During configuration (installation), the evaluation program creates a symbolic link between the file in the *file* field and the CLODB of the identified cluster with the same pathname. The absolute pathname of the file in the *file* field is appended to the /var/clos/*clos_name*/root home directory of CLODB. The resulting symbolic link is permanent, i.e. it is not deleted when cod_prototype(1M) is called with the –r (remove) option.

file Pathname of the file used to execute the operation selected in the *flags* field. This pathname must be absolute, otherwise the entry is ignored by the evaluation command.

args If the *flags* field requests the execution of a command, this field is transferred as an argument string. The string can also contain spaces.

If the *flags* field contains the c character, the file characteristics are specified here. The format is then:

mode owner group

If the values cannot be established, the default values for *mode*=755, *owner*=root and *group*=sys are set.

COMMANDS

cod_prototype(1M)
This command is used to evaluate the COD_prototype files defined by the applications. As an argument to the command, it is specified whether a new CLODB is to be configured (–i option), whether a CLODB is to be removed (–r option) or whether CLODB is to be updated following the removal of a node from the cluster (–u option). For a subsequent installation, an application should invoke cod_prototype(1M) with the –i option.

Warning: The command is only effective locally on a node.

clos_adm(1M)
This command sets up a new CLOS name space. Afterwards, a cod_prototype evaluation (with the –i option) is also performed automatically for this new name space.

node_rm(1M)
When a node is removed from the cluster, the COD_prototype files are evaluated before CLODB is compared with the cod_sync(1M) call. CLODB (–u option) is updated on the remaining nodes in the process and the –r option is used on the node to be removed.

Note:

No master node is used for managing CLODB. A cluster-wide update is not envisaged since it is assumed that such updates do not happen very frequently and are also not performed in parallel on several nodes. An update only becomes effective throughout the cluster when the cod_sync(1M) command is issued. Automatic synchronization, performed at regular intervals, is not planned.

FILES

/opt/COD/prototypes

/var/clos/*clos_name*/root

SEE ALSO

uname(1), clos_adm(1M), cod_prototype(1M), cod_sync(1M), node_add(1M), node_rm(1M), clodb(7), clos(7).

Fujitsu Siemens Computers Add-on Reference Manual

draid(4)

NAME

`draid` - database for the daemon that monitors both channels of aRAID box (Dual RAID Channel Daemon)

DESCRIPTION

The file `/etc/draid` contains a list of RAID boxes and their controllers to be used by `draidd(1M)` . Each entry for a RAID box is an individual line in the following format:

Dual-RAID-name : □
Number-of-controllers : □
First-controller-device-node : *LUNs-of-first-controller* : □
Second-controller-device-node : *LUNs-of-second-controller* :

The fields must be separated by ":" (colon). A "#" (hash sign) in the first column of a line denotes a comment line. All characters to the end of the line are not interpreted by functions that read this file.

Dual-RAID-name
The name of a valid virtual RAID device. This name is used in subsequent accesses to the RAID box.

Number-of-controllers
The number of controllers linked to the RAID box. Two controllers are supported at present.

First-controller-device-node
The name of the first physical host controller which is connected to a RAID box controller.

LUNs-of-first-controller
The LUN(s) to be used by the first controller. The various different LUN(s) are given as decimal numbers separated by blanks.

Second-controller-device-node
The name of the second physical host controller which is linked to a RAID box controller.

LUNs-of-second-controller
The LUN(s) to be used by the second controller. The various different LUN(s) are given as decimal numbers separated by blanks.

EXAMPLES

`draid0 : 2 : ios0/sraid030 : 1 2 : ios0/sraid051 : 0 3`
The RAID box is connected to the host over the host controllers `/dev/ios0/sraid030` and `/dev/ios0/sraid051`.

The host controller `ios0/sraid030` uses the LUNs one (1) and two (2).

The host controller `ios0/sraid051` uses the LUNs zero (0) and three (3).

Applications can access this RAID box over the device nodes `/dev/draid/draid0l`*LUN*`s`*PARTITION*. In this case, *LUN* must be a number between zero and three and *PARTITION* must be a number between zero and 15.

For example, `/dev/draid/draid0l0s0` (LUN: 0, Partition: 0) must be mounted after `/mnt0`, and `/dev/draid/draid0l1s2` (LUN: 1, Partition: 2) must be mounted after `/mnt1`. Each access to the file system mounted on `/mnt0` uses `/dev/ios0/sraid051` as an access controller. If, for any reason, `ios0/sraid051` should fail, `draidd(1M)` then directs any further access to `/mnt0` to the controller `ios0/sraid030`, without involving the system administrator.

NOTES

`draidd(1M)` reads and interprets `/etc/draid` at startup or after it receives the hangup signal (`SIGHUP`). When `draidd(1M)` is started the controller is not configured, as described in `/etc/draid`, by default. However, the current configuration is read out from the RAID box controller and the system driver is set with these values. This configuration may deviate from that described in `/etc/draid`. However, `draidd(1M)`

requires *Dual-RAID-name* to enable access to the RAID box. Should `draidd(1M)` receive a `SIGHUP` signal, it attempts to configure the RAID box, as described in `draid`.

The start procedure is required if a controller is defective at startup, and the corresponding RAID box was reconfigured during a previous run of `draidd(1M)`. It is also possible in a multihost environment that a RAID box was configured by another host. In this case, the actual host must assume the configuration of the removed host.

FILES

`/etc/draid`

SEE ALSO

`draidd(1M)`, `draid(7)`.

Fujitsu Siemens Computers Add-on Reference Manual

hvgdstartup(4)

NAME

`hvgdstartup` - Reliant Monitor generic detector startup file

SYNOPSIS

det_name –k*K* –t *T* [–d]

DESCRIPTION

The `hvgdstartup` file contains lines controlling the generic detector startup process. The detector time period is also specified in the `hvgdstartup` file. Each generic detector is started as a separate UNIX process, executed from a separate binary file.

OPTIONS

det_name
: The generic detector executable name.

–k*K* Assign a unique `rKind` resource type as *K* (from 0 to 63) to a unique resource. This number must match a defined generic resource `rKind` attribute.

–t*T* Define a time period *T* (in seconds) that Reliant Monitor waits before calling `chk_gdetector()`.

–d Force debugging information to be printed in the *det_name*`log` file in the logfile directory.

EXAMPLES

The following sample lines will allow `hvcm` to start detectors `DBdetector` and `VideoCard` as kind 0 (`k0`) and kind 4 (`k4`), respectively:

```
DBdetector –t5 –k0
VideoCard  –t4 –k4
```

SEE ALSO

`hvgdmake`(1M) .

Fujitsu Siemens Computers Add-on Reference Manual

mc2tab(4)
SIDLM

NAME

`mc2tab` - MC2 module configuration file

DESCRIPTION

The MC2 module is a protocol converter that is pushed on top of network drivers like LCE to make them accessible under the MAC2 interface. The MAC2 interface is used by the Distributed Lock Manager (DLM) and the MAC Switch Driver (MSW).

The `/etc/mc2tab` file contains one line for each interface to be accessed under MAC2. Each line contains the name of a network interface upon which the MAC2 module is to be pushed. The format for the name is the concatenation of the fields "ctype" and "log" as shown by the `showconf(8)` command on the RM600 or in general the node name in `/dev/dlpi/`. `/etc/mc2tab` is read by the network drivers startup scripts like `/etc/init.d/lce`.

EXAMPLES

The following example of an `/etc/mc2tab` file makes the network interfaces `lce2` and `lce3` accessible under MAC2:

```
lce2
lce3
```

FILES

`/etc/mc2tab`
MC2 module configuration file

SEE ALSO

`clustertab(4)`, `mswtab(4)`, `dlm(7)`, `mswconfig(8)`.

mswtab(4)
SIDLM

NAME

`mswtab` - MAC switch driver configuration file

SYNOPSIS

`sw` *N* [*number*] [*network_address*] [*interface_name*] [*unit_number*]
[*interface_name*] [*unit_number*]

DESCRIPTION

The `/etc/mswtab` file describes the MAC switch driver configuration. It is used by the `mswconfig(8)` command to administer the MAC switch driver.

The following list describes the fields of the command line:

`sw`*N* Name of the MAC switch driver. *N* is an integer from 0 to 9.

number
Number of Ethernet interfaces that comprise a single MAC switch driver virtual interface.

network_address
Network address on this node for the Virtual Ethernet network. Usually, it should be set to the physical Ethernet address of the first interface. To obtain this physical address, use the command `etherstat(1M)` .

interface_name
Name of the physical network interface. It corresponds to the `ctype` field of the output of the `showconf(8)` command for the LAN controller used. This is equivalent to the device name in `/dev/mac`. Currently, one possible value is `lce` for the Ethernet controller LCE2.

unit_number
Unit number of the physical network interface. It corresponds to the field "log" of the output of the command `showconf(8)` for the Ethernet interface used. It is also the number extracted from the nodename of the Ethernet interface in `/dev/dlpi`.

One or more virtual Ethernets can be specified in `/etc/mswtab` by using different names `sw`*N*.

EXAMPLES

The following example of an `/etc/mswtab` file is for a MAC switch driver virtual interface `sw0` with two physical pieces, `lce1` and `lce4`.

```
sw0  2       8:0:6:5:51:01
        lce     1
        lce     4
```

FILES

`/etc/mswtab`
MAC switch driver configuration file

SEE ALSO

`ifconfig(1M)`, `etherstat(1M)`, `clustertab(4)`, `mc2tab(4)`, `strcf(4)`, `mswconfig(8)`, `showconf(8)` .

Miscellaneous Facilities(5)

observe(5)

NAME

`observe` - high-availability monitor

DESCRIPTION

The high-availability monitor OBSERVE running on our Reliant UNIX platforms provides a high level of business continuance capabilities for mission-critical open systems applications.

High-Availability OBSERVE Clusters

High-availability clusters consist of redundant servers plus redundant hardware components and redundant data storage devices. In case of a failure of one component applications using the failed component can be continued on the corresponding redundant counterpart. The hardware is monitored by our high-availability monitor OBSERVE, which detects failures and initiates recovery procedures (e.g. in case of a server failure the failover of the applications to the other host). An OBSERVE cluster can be customized to fit your special requirements (see configuration files).

PACKAGES

OBSERVE
: The high-availability monitor OBSERVE

OBSMAN
: Man pages of the OBSERVE commands

OBSMON
: OBSERVE graphical user interface

OBSNET
: TCP/IP support in OBSERVE clusters

OBSSCON
: Single console for OBSERVE clusters

OBSSRDF
: EMC²-SRDF support in OBSERVE clusters (see `/opt/observe/doc/OBSSRDF/obssrdf_manual.*`)

COMMANDS AND MAN PAGES

`obsallow`(1M)
: allow OBSERVE start

`obsforbid`(1M)
: forbid OBSERVE start

`obsstate`(1M)
: define or change a computer's OBSERVE status

`obsreturn`(1M)
: reestablish normal operation after failover

`obschange`(1M)
: exchange original and backup computer functions

`obsobject`(1M)
: control object monitoring

`obsinfo`(1M)
: display system information

`obsobjinfo`(1M)
: display object information

`obsdown`(1M)

shut down computer or terminate OBSERVE

`obshost`(1M)
: enable or (temporarily) disable system failover

`startS98`(1M)
: start OBSERVE

`obsrestart`(1M)
: restart failed observer

`lan_host`(1M)
: retrieve information about the OBSERVE status

`obsconfig`(1M)
: compile the OBSERVE configuration file

`dumpfile`(1M)
: decode and display binary configuration file

`obsress`(1M)
: start or stop resources

`oswisusi`(1M)
: actuate or query SUSI switches

`oswitaclan`(1M)
: switch TACLAN devices

`oswiscu2`(1M)
: actuate or query SISO or SCU switches

`oswiraid`(1M)
: switch RAID boxes or EMC² Symmetrix DRAID

FILES

`/opt/observe/reactions/usr-samples`
: This directory contains the standard reaction scripts for resource handling and several corresponding readme files.

`/opt/observe/reactions/usr-samples/README.usr-samples`
: This file contains explanations about the structure of reaction scripts and how to integrate a script into the OBSERVE system. Resources which do not have their own readme file are explained here as well.

`/opt/observe/reactions/usr-samples/README.*`
: Readme files for reaction scripts. The following are available:

Reaction script	Readme file
`000draid`	`README.draid`
`010gmirror`	
`020filesystem`	
`021fuserkill`	
`022share`	
`023nfs_filesystem`	`README.nfs_filesystem`
`025ip`	`README.ip`
`025obsnetif`	
`026obsnetdied`	
`030database`	`README.informix`
`040spool`	`README.spool`

```
040xprint          README.Xprint
045networker       README.networker
050application
999wreset          README.wreset
```

`/opt/observe/reactions/usr-samples/param.usr`
OBSERVE resource parameter file example.

`/opt/observe/konfig/config-sample.11`
OBSERVE configuration file example.

`/etc/default/observe`
OBSERVE default parameter file.

SEE ALSO

User's Guide *OBSERVE*.

Protocols(7)

cens(7)
Cluster Operating System

NAME

`cens` - introducing the `CENS` Cluster Event Notification Service

DESCRIPTION

The `CENS` Cluster Event Notification Service is a subsystem of the Cluster Operating System `CLOS` [see `clos(7)`] and is implemented using two daemon processes.

The `Emitter` daemon [see `emitter(1M)`] is started on every node for every event type or event message class. It sends the event messages to one (or more) `Collector` daemons [see `collector(1M)`] which, in turn, pass them on to a target file or call a command with the message as its argument.

The `event_start(1M)` , `event_stop(1M)` and `event_adm(1M)` commands are available to perform administration tasks as well as for starting, ending or listing daemons.

`CENS` can be used to group event messages into various classes or types. Each class is assigned a name. The type or class of message is specified using the following parameters:

Name of the event class
This name can be passed as an argument to the commands (`-t` option) and the remaining parameters derived from it.

Source file
This is the name of the file from which the event messages on the individual nodes are taken. This file can be a normal file, a FIFO or a special device file (e.g. `/dev/tty`). The `Emitter` daemons read this file line by line and send each line (message) to the `Collector` daemons. If the *source file* is a normal file, it is converted to a FIFO when it has been fully read and the system then waits for further incoming messages.

Target file
This is the name of the file in which the `Collector` daemon stores the event messages that arrive via the Cluster Interconnect or which it executes as a command. This file is always a normal file for the `Collector` daemon.

Control symbol
This is a symbol that determines how the `Collector` daemon is to interpret the *target file*. The `p` symbol means that the event messages are stored in the *target system*. The `c` symbol means that the

target file is executed as a command with the message as a single argument. The `r` symbol can be used to reserve a marked channel with a name. `CENS` is not started for classes with the `r` control symbol.

User rights

The execute or access rights to the *target file* are derived from this parameter. It is a `$LOGINNAME` from `/etc/passwd`.

Marking channels

The `CENS` daemon processes use the services of the `Closmon` monitor to enable them to exchange data via a Cluster Interconnect [see `closmon`(1M)]. This provides them with private channels for every cluster node. The channels are marked so that the monitor can distinguish between the private channels of a subsystem. Since separate channels can be used for all classes of events, a flag must be defined for every class.

If the name of the event class is specified as an argument in `CENS` commands, the necessary parameters are taken from the `/etc/default/cens` file. This file contains the event classes defined by an administrator with the `event_adm`(1M) command. An entry in this file corresponds to a class (type) of event message and has the following format:

CLASS_NAME FLAG OWNER KEY SOURCE_PATH DEST_PATH

Example:

For historical reasons, the `HWERRORS` event class is always defined. The source data for the `Emitter` daemons must be taken from the `/var/spool/SIconf/hwerrors` event files on every node. What is new for this event class is the *key*=6846322 definition (decimal number of ASCII characters `hwr`) and the `/var/spool/SIconf/global_hwerrors` target file that is shared by all nodes. The entry has the following format:

```
HWERRORS p root 6846322 /var/spool/SIconf/hwerrors \
/var/spool/SIconf/global_hwerrors
```

The `event_start`(1M) command is used to configure the monitoring of a class of event messages. An `Emitter` daemon, which evaluates the source file and waits for further messages that are written to this file, is started on every integrated node in the cluster. A `Collector` daemon, which writes every individual message to the target file, is started on the local node.

If, in the meantime, a new node is added to the cluster group, a suitable `Emitter` daemon is automatically started on it by `CENS`. This daemon passes on its messages to the `Collector` daemon. If a node is removed from the cluster group, the relevant daemon is also taken from the `Emitter-Collector` network. If the last `Collector` daemon is to be removed when removing a node, monitoring of the event class is terminated.

A lock file in the `/var/clos/`*clos_name*`/locks` directory prevents an `Emitter` daemon from being started twice on every node. The name of the lock file comprises the daemon type (emitter) followed by an underscore and the flag used to identify the event class (*key* class parameter). The file contains the process number of the currently active daemon.

Since it is permissible for several `Collector` daemons of the same event class to be active on a single node, they are only prohibited from writing to the same target file (or executing the same command). The idea behind the lock strategy is that when starting a `Collector` daemon in the `/var/clos/`*clos_name*`/locks` directory, a subdirectory called `collector_`*key* is created for every event class in which every daemon creates a file bearing the name of its process number. The name of the target file is written to this file and a check is carried out each time the `Collector` daemon is started to determine if there are any conflicts. Only one `Collector` daemon is started for a target file in every event class.

The `Emitter–Collector` network is only stopped with the `event_stop`(1M) command.

DIAGNOSTICS

Considering that the event messages are processed by daemon processes that do not have any controlling TTY channels for outputting error messages, the messages are written to files stored in the `/var/clos/errmsg` directory. The names of these files are made up of the type (collector or emitter), followed by an underscore and the flag assigned to the event class (*key* class parameter). The daemon

messages include the time, date and process number, and are appended to the corresponding file. Thus, every event class has a dedicated log file which records when the daemon started and ended in addition to any errors that may have occurred.

FILES

`/etc/default/cens`
`/etc/passwd`
`/var/clos/errmsg`
`/var/clos/`*clos_name*`/collector`
`/var/clos/`*clos_name*`/emitter`
`/var/clos/`*clos_name*`/locks`

SEE ALSO

`collector`(1M), `emitter`(1M), `event_adm`(1M), `event_start`(1M), `event_stop`(1M).

Fujitsu Siemens Computers Add-on Reference Manual

clodb(7)
Cluster Operating System

NAME

clodb - introducing the CLODB Cluster Object Database

DESCRIPTION

The CLOS Cluster Operating System [see clos(7)] requires data on all cluster nodes from which it can obtain its administration information. This data is stored in one or more files. Since CLOS has equal rights on all nodes, these files must be directly accessible for all nodes. This may result in problems with consistency since every node has copies of these files. The set of data and files required by CLOS is known collectively as the Cluster Object Database (CLODB). Distributing and maintaining the consistency of these files is the most important task performed by the CLODB subsystem.

STRUCTURE

A /var/clos/*clos_name*/root root directory exists on all nodes in a cluster called *clos_name*. The remaining CLOS subsystems can store any file trees in this directory. The names, owners and structures of the files are transparent for the CLODB subsystem. It only has to ensure that identical copies of this file tree are distributed on all nodes in a cluster. The total number of file trees located under the root directory represents the *installed* Cluster Object Database. In addition, to ensure the consistency of the installed CLODB, every *clos_name* cluster node contains an archive called /var/clos/*clos_name*/COD, in which the entire installed CLODB is frozen as an archive. The modification date for this COD archive is kept the same on all nodes. CLODB is said to be consistent if the contents of the COD archive are identical with the CLODB installed under the root directory. You can use the codadm(1M) command to check the consistency, create a COD archive or install a CLODB.

CONSISTENCY

A system administrator can change or reconfigure files locally that belong to CLODB. However, such modifications only affect the local node initially. In general, it is not possible to stipulate the commands that the system administrator can use to update the files in CLODB. For this reason, the /var/clos/*clos_name* home directory of the *clos_name* cluster contains a file called modify which must always be updated using touch if any file in CLODB is changed. This should be carried out either explicitly by the system administrator or implicitly using CLOS commands. If the date of the modify file is more recent than that of the COD archive, the system administrator (or programs invoked to perform a consistency check) knows that the installed CLODB is no longer distributed consistently over the nodes. The system administrator can now opt to restore the existing CLODB using an Install command [see codadm(1M) , –i option] or, if the modification applies to the entire cluster, to distribute it using cod_sync(1M) . When both commands have concluded, the COD archive and the modify file have the same date once again.

To ensure that the COD archive is not manipulated by commands that do no belong to CLOS, a flag is set in the modify file when generating the COD. Setting a flag in this case involves entering all CLODB files in the modify file along with their checksum and block size. A separate command checks the installed files and their checksums [see codadm(1M) , –e option].

An automatic consistency check is generally only performed when a new cluster is configured [see clos_adm(1M)]. In this case, the closmon(1M) database /etc/saf/*clos_name*/_pmtab is integrated in CLODB as a symbolic link and the COD archive is generated.

Every time a connection setup request is received from the Closmon monitor, the clserver(1M) Cluster Service daemon responds with the date of the COD archive in the respective cluster. The monitor compares this with the date of its local archive and reports the discrepancy to the system administrator on request [cod_check(1M) command]. The system administrator must then decide whether it is necessary to synchronize these dates and which CLODB should be distributed.

COMMANDS

A number of commands have been provided in order to edit CLODB that are part of CLOS. All of these commands are stored in the /var/clos/bin directory and are also symbolically linked in the command directories of the Reliant UNIX operating system when the SIclos package is installed. The following CLOS commands are available for updating CLODB:

codadm(1M)
: This is used to manage the locally installed CLODB and the COD archive and comprises the following functions:

  Marking a CLODB update (–m – modify)
  : When a CLODB file has been updated, this command is used to create a new list of the files belonging to CLODB along with their checksums in the modify file. Only real files and symbolic links feature on this list. Directories are ignored. The modify file is given the current date.

  Checking the consistency of the local CLODB (–e – evaluate)
  : This is used to check the consistency of an installed CLODB using the existing COD archive on the local node. The command reads the modify file from the COD archive. The first task it performs is to compare the date of the modify file with that of the COD archive. Any discrepancy between dates is reported. In this event, a new checksum file is created and then compared with that in the archive. All differences between files are displayed.

  Creating a (partial) archive (–c – create)
  : All files located in the installed CLODB are written to the archive. Symbolic links are also identified and their sources recorded in the archive. A new modify file is created for marking purposes and is recorded in the archive. After the command has concluded, the date of the modify file is identical to that of the COD archive.

  Installing a (partial) archive (–i – install)
  : The archive is installed as a file tree on the node. In this case, the modify file receives the same date as the COD archive, and the CLODB is consistent on the local node. The tree that exists in the COD archive is the one that is installed as CLODB. Files that were changed locally before this installation are overwritten and files that were created in CLODB but not archived disappear. The resulting CLODB is consistent.

cod_sync(1M)
: This command allows the system administrator to determine that the locally installed CLODB is exported to all nodes in the cluster. A new COD archive is created before CLODB is exported. Any nodes that were not integrated when the cod_sync command was called, have their CLODB updated the next time they log on to the cluster.

SYNCHRONIZATION

Using the special cod_sync(1M) command, a consistent CLODB is created locally and the COD archive is distributed to and installed on all nodes in the cluster. If any nodes are not available when the cod_sync command is called, the recovery mechanism assumes responsibility for subsequent distribution [see clos(7)].

Controlled by options, only the part of CLODB that has changed can be stored in a partial archive [see codadm(1M) , –p option] and then distributed with the cod_sync(1M) command, –p option. This is particularly useful if the time taken to transfer the entire archive is excessive in relation to the time it takes to install the archive.

In order to allow the subsystems of the Cluster Operating System to carry out a number of measures before CLODB is installed, all executable command files located under /var/clos/*clos_name*/preinstall are executed. Similarly, all command files under /var/clos/*clos_name*/postinstall are executed when the archive has been installed. The name of a file in which all installed files are listed line by line is specified as an argument for each of these command scripts. Each subsystem can therefore decide which measures are to be adopted before and after installation.

Note:

A master node is not used to manage CLODB. It is not intended to have cluster-wide notification of updates

since it is assumed that such updates do not occur very frequently nor are they performed in parallel on several nodes. Any cluster update only becomes effective after `cod_sync(1M)` has been executed. Regular, automatic synchronization is not envisaged.

FILES

`/var/clos`

`/var/clos/`*clos_name*`/COD`

`/var/clos/`*clos_name*`/locks`

`/var/clos/`*clos_name*`/modify`

`/var/clos/`*clos_name*`/postinstall`

`/var/clos/`*clos_name*`/preinstall`

`/var/clos/`*clos_name*`/root`

`/var/clos/`*clos_name*`/recover/`*tag_name*

SEE ALSO

`clos_adm(1M)`, `closmon(1M)`, `clserver(1M)`, `codadm(1M)`, `cod_check(1M)`, `cod_sync(1M)`, `clos(7)`.

Fujitsu Siemens Computers Add-on Reference Manual

clos(7)
Cluster Operating System

NAME

clos - introducing the CLOS Cluster Operating System

DEFINITIONS

Cluster
: In the context of the CLOS Cluster Operating System, the term cluster refers to a set of UNIX systems (nodes) which form a management unit and are connected to one another by a communication medium. The communication medium (Cluster Interconnect) must enable direct communication between every two nodes in a cluster via an IP protocol [see tcp(7) and udp(7)]. Each cluster is assigned a name *clos_name* since a node can be simultaneously integrated in several clusters. This name is assigned to the relevant node with the clos_adm(1M) command and the node, in turn, is assigned to a cluster with the node_adm(1M) command.

  All CLOS commands are only valid in one cluster. The name of the cluster can be specified for the commands with the

  - –C option
  - the $CLOSNAME environment variable [see environ(5)]
  - or the default setting in the /var/clos/.CLOSdefault file

  Note:

  A CADMIN cluster name is assigned when a CLOS is installed and is in turn installed by default on every node. The configuration software can now assign nodes to this cluster. This CADMIN cluster is set as the default for CLOS commands. The configuration software can insert nodes in this cluster and then define and configure further clusters within this cluster. Such additional clusters within the CADMIN management cluster can include nodes that do not use a public LAN as a Cluster Interconnect, rather a private network regardless of how it is started (e.g. the fail-safe ICF).

Tag names
: Tag names are the names of the nodes as defined by the cluster management for a cluster. While tag names may differ from the Internet names or node names of a system, we recommend that you keep the tag names identical with the node names [see uname(1)]. The command used to set this name is node_adm(1M) .

Node channel
: By node channel, we mean a FIFO, which is mapped from the local node to another node in the cluster. These FIFOs are mapped by the Closmon monitor [see closmon(1M)] and are created in the /dev/clos/*clos_name* directory with the tag name of the respective node. If the files in the /dev/clos/*clos_name* directory are regular files, these nodes are still configured in the cluster but cannot yet be accessed.

Remote nodes
: We refer to a remote node if the node we are currently working on can communicate with another node in the cluster. Each of these other nodes is called a remote node. The relevant file in /dev/clos/*clos_name* is a FIFO.

DESCRIPTION

The Cluster Operating System CLOS is an add-on product on Reliant UNIX systems. It works within a cluster in the context of the above definition and consists of a set of commands that simplify the task of managing, configuring and maintaining a group of nodes. It also comprises the following subsystems:

- Communications subsystem
- Node Management subsystem

- Diagnostic subsystem
- Cluster Event Notification subsystem
- User Administration subsystem
- Cluster Object Database management

Since daemon processes are started for some services for every cluster configured with clos_adm(1M) , the problem reports issued with these daemon processes are saved in the /var/clos/errmsg directory to a file bearing the name of the daemon, possibly with an argument ID. The CLOS communication system daemons store their messages for each cluster in the /var/saf/*clos_name* directories while the global CLOS server daemon stores its messages in the /var/saf/clos_service directory.

To exclude the possibility of several daemons being started for each cluster, lock files are created in the /var/clos/*clos_name*/locks directory bearing the name of the daemon and a specific ID. The process number of the respective process is entered in these lock files.

SUBSYSTEMS

CLUSTER COMMUNICATION

The Closmon monitors [see closmon(1M)] are used on every node to enable all nodes in a cluster to communicate with one another. Using inetd(1M) , the monitors initiate service processes, known as cluster services [see clserver(1M)]. The interaction between closmon and clserver is determined by the cluster communication system.

For every node configured in the cluster, the Closmon monitor creates a file with the node tag name in the /dev/cluster/*clos_name* directory. If the node is integrated in the cluster, the normal file is converted to a FIFO. These FIFOs can use the CLOS subsystems as so-called "node channels" for communication purposes.

The CLOS commands and daemons use the node channels privately for their services. They can assign specific characteristics to the channels for this purpose. You can mark a channel [see set_chan_key(3-clos)] or query the status of the remote station of the (marked) channel [see set_chan_ctrl(3-clos)].

Similarly, using an open node channel, a process can issue a command that is executed in the remote station node [see chan_cmd(3-clos)]. The options offered by the processes on the node channels are transmitted in a separate log to the Closmon monitor with high priority messages [see putmsg(2)].

The remote station responses are transferred to a process using CLOS control messages. These are high priority messages which, in addition to a type, contain a data area for displaying the results of preceding control functions. This data area is the struct ctrlresp structure with the following elements:

```
resp_id;
resp_val;
```

The following types of CLOS control message are available:

CLOS_OPEN
: This is the response to a node channel in which the set_chan_ctrl(3-clos) control function is used to signal the status for the remote station. resp_val specifies the number of open channels with the same flag on the remote node. The resp_id is 0.

CLOS_CLOSE
: This is the response to a node channel in which the set_chan_ctrl(3-clos) control function is used to signal the status for the remote station. The resp_val and resp_id structure elements are 0. This signals that there no channel with this flag is still open on the remote station.

CLOS_ACKN
: A command was issued on the node channel for nodes on the remote station with the chan_cmd(3-clos) function. With this message, the CLOS communication system signals that the command has been accepted on the remote node. The resp_val structure element contains the error number if transmission of the command failed. If the value is 0, the command

is executed and no error occurs when transferring the command. The resp_id structure element contains the command ID set by the requester.

CLOS_RETURN
A command was issued on the node channel for the nodes on the remote station with the chan_cmd(3-clos) function. With this message, the CLOS communication system signals that the command has been accepted on the remote node. The resp_val structure element contains the end status for the command as described in wstat(5). The resp_id structure element contains the command ID set by the requester.

CLOS_RETPID
The remote station indicated its process ID and a numeric value with a chan_send_pid(3-clos) function. This message type contains the process ID in the resp_id element and the transmitted numeric value in the resp_val element.

CLOS_SENDVAL
The remote station closed its node channel for the local node with the same ID with the chan_close(3-clos) function. This CLOS control message type contains the process ID in the resp_id element and the transmitted numeric value for the sending process in the resp_val element.

CLOS_RETOPEN
This CLOS control message type is the response to a chan_get_open(3-clos) process function call. The resp_val element contains the number of channels currently open with the same ID for this node. The ID is contained in the resp_id element.

CLOS_RETVAL
This is the acknowledgement sent by the Closmon monitor indicating that the values of a chan_close(3-clos) job have been issued. The user is informed of the type in the form of a return value. The message itself is a copy of the job.

CLOS control messages are sent *spontaneously* if

- a channel with the same ID is open on the remote station
- a channel with the same ID is closed on the remote station
- a command reached the remote station
- a command was terminated on the remote node

All other CLOS control messages are a direct result of a node_channel(3-clos) function.

CLUSTER EVENT NOTIFICATION SERVICE
The CENS Cluster Event Notification subsystem [see cens(7)] is implemented by two daemon processes.

The Emitter daemon [see emitter(1M)] is started for every event on every node. It sends the event messages to one (or more) Collector daemons [see collector(1M)] which, in turn, pass these on to a target file. Administration tasks such as starting, ending or listing daemons are performed with the event_start(1M), event_stop(1M) and event_adm(1M) commands. The assignment of the individual event messages to a class is described in the /etc/default/cens file.

CLUSTER OBJECT DATABASE
The CLODB Cluster Object Database [see clodb(7)] is used to manage the cluster-specific data. The database itself is managed using the codadm(1M) command. It is important with CLODB that the data be duplicated consistently on all nodes in the cluster. The cod_sync(1M) command can be used to ensure this. A CLODB is available as an /var/clos/*clos_name*/COD archive file and can be installed using the codadm command. A CLODB is always installed under the /var/clos/*clos_name*/root directory.

LIBCLOS
A library with various functions is offered to support programming of commands and daemons as CLOS subsystems. These functions can be found in the /usr/lib/libclos.a archive.

RECOVERY MECHANISM

CLOS supports a generic recovery mechanism that is used particularly when synchronizing CLODB with the cod_sync(1M) command.

Every time a node signs on to another Closmon monitor, this monitor searches the local /var/clos/*clos_name*/recover/*tag_name* directory for shell command scripts. If any script is found, it is called with the *tag_name* argument. If the script ended successfully with a return value of 0, the script itself is deleted from the directory. It is worth noting that the script is called locally and can be executed on another node in parallel with a recovery script.

If a subsystem recognizes by a node failure that a recovery is necessary, it must place the script in the directory of the respective node for which it is to be executed when this node can be accessed again. If the script was distributed to all accessible nodes, it is executed by all of these nodes the first time the failed node signs on to them. This can result in parallel or successive script calls.

If a recovery script is not distributed, it can only be executed once. There is always the possibility that the actual node containing the script could not be accessed when reintegrating the failed node. The following safeguards must therefore be put in place:

1. Critical data accesses must be protected through mutual exclusion.
2. A call that is executed sequentially a number of times must always have the same result.
3. If the script recognizes that it has already been executed on another node, it should terminate with the return value 0 so that it is then deleted from CLOS.

Following execution on a node, a *script_name* recovery script must be removed on the other nodes which are not currently integrated. To ensure this happens, create a *script_name* file in the /var/clos/*clos_name*/recover/*tag_name* directory containing all the tag names to which the script was distributed. CLOS then removes the script when reintegrating the relevant node.

DIRECTORIES

The home directory for the Cluster Operating System CLOS is /var/clos. The following fixed directories were created specially for all CLOS cluster incarnations:

bin Contains all commands of the CLOS Cluster Operating System.

default
: Contains files that are copied with every incarnation to their CLODB.

errmsg
: Contains files with error messages from the individual daemon processes.

lib Contains everything that a programmer requires: header file and the libclos.a library.

In addition, a directory called *clos_name* is created for every CLOS cluster to indicate the name of the cluster.

The home directory of every cluster incarnation is /var/clos/*clos_name* and has the following subdirectories:

nodes
: Contains the description of the status changes in the cluster in the clos_state file, as well as a *tag_name* subdirectory (node tag name) for every configured node containing the system file for the system information and state file for the node status. In addition to the system and state files for the local node, the this subdirectory contains the tag name of the local node in the tag file. The presence of a subdirectory with the tag name indicates that the node is configured in the cluster.

tmp Auxiliary directory for a number of commands.

collector
: Directory for log and status files for Collector daemons in the CENS subsystem.

emitter
: Directory for log and status files for Emitter daemons in the CENS subsystem.

locks
: Directory under which various commands create lock files [see clos_lock(1M) and clos_unlock(1M)].

`root` Home directory for the installed `CLODB` Cluster Object Database.

`preinstall`
Directory for scripts that must be executed before `CLODB` is installed.

`postinstall`
Directory for scripts that must be executed after `CLODB`is installed.

`recover`
Directory for storing scripts. If a command is to be executed for a node that currently has the status `NOT_AVAIL`, this directory is used. The user must save the command to be executed under any name in a subdirectory of `recover` with the tag name of the failed node. If the node status changes to `INTEGRATED`, the `Closmon` monitor searches the relevant subdirectory and executes any scripts it finds there.

In addition to the above-named directories, the following directories are available for every cluster incarnation for log and trace files of the `CLOS` `clserver`(1M) and `closmon`(1M) `CLOS` daemons:

`/var/saf/`*clos_name*
All trace, status and log files of the `Closmon` monitor are created here.

`/var/saf/clos_service`
All trace, status and log files of the cluster service daemon `clserver`(1M) are created here.

FILES

`/dev/clos/`*clos_name*

`/dev/clos/`*clos_name*`/`*tag_name*

`/var/clos`

`/var/clos/.CLOSdefault`

`/var/clos/bin`

`/var/clos/errmsg`

`/var/clos/default/cens`

`/var/clos/default/_pmtab`

`/var/clos/lib`

`/var/clos/lib/cluster.h`

`/var/clos/lib/libclos.a`

`/dev/clos/`*clos_name*

`/var/clos/`*clos_name*`/COD`

`/var/clos/`*clos_name*`/collector`

`/var/clos/`*clos_name*`/emitter`

`/var/clos/`*clos_name*`/locks`

`/var/clos/`*clos_name*`/modify`

`/var/clos/`*clos_name*`/nodes`

`/var/clos/`*clos_name*`/nodes/`*tag_name*

`/var/clos/`*clos_name*`/nodes/`*tag_name*`/state`

`/var/clos/`*clos_name*`/nodes/`*tag_name*`/system`

`/var/clos/`*clos_name*`/nodes/this`

`/var/clos/`*clos_name*`/nodes/this/system`

`/var/clos/`*clos_name*`/nodes/this/state`

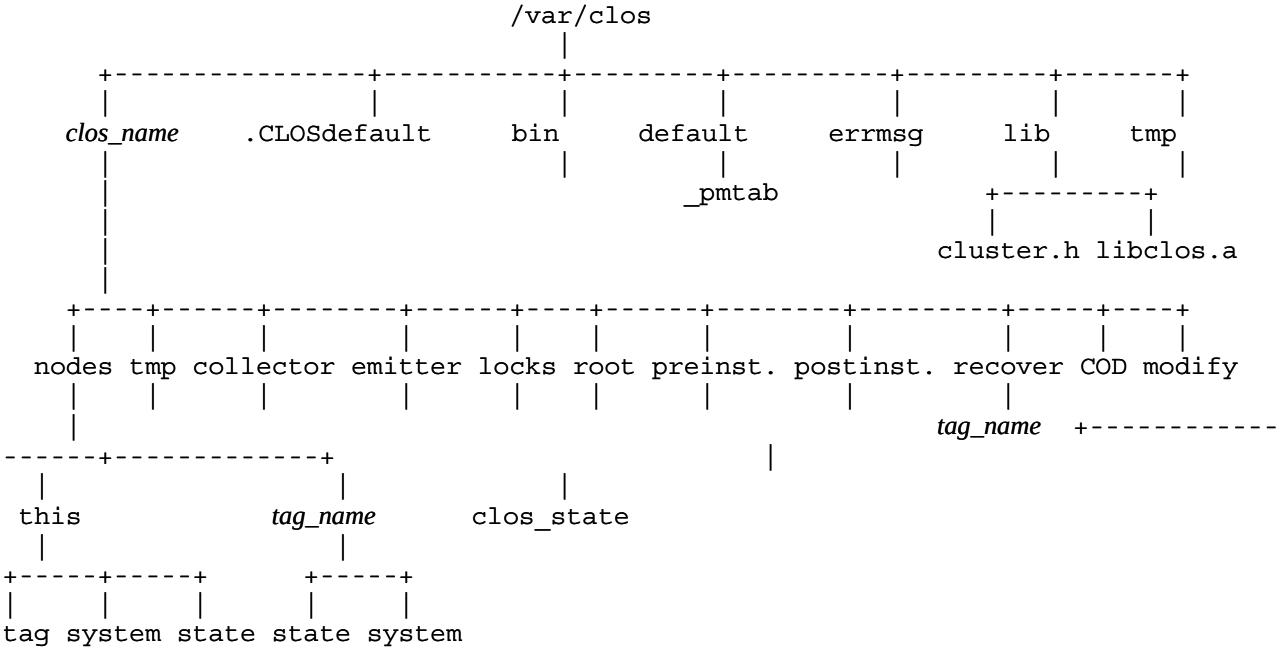
`/var/clos/`*clos_name*`/nodes/this/tag`

`/var/clos/`*clos_name*`/postinstall`

`/var/clos/`*clos_name*`/preinstall`

/var/clos/*clos_name*/root
/var/clos/*clos_name*/recover
/var/clos/*clos_name*/recover/*tag_name*
/var/clos/*clos_name*/tmp
/var/saf/*clos_name*
/var/saf/clos_service
/etc/default/cens
/etc/saf/*clos_name*
/usr/lib/libclos.a

The complete tree structure of a CLOS Cluster Operating System under /var/clos is represented as follows:



SEE ALSO

closmon(1M), clserver(1M), codadm(1M), collector(1M), emitter(1M), event_adm(1M), inetd(1M), getmsg(2), putmsg(2), chan_cmd(3-clos), chan_close(3-clos), chan_get_open(3-clos), chan_send_pid(3-clos), set_chan_ctrl(3-clos), set_chan_key(3-clos), passwd(4), wstat(5), cens(7), clodb(7).

Fujitsu Siemens Computers Add-on Reference Manual

commd(7)
SIDLM

NAME

commd:
commdenv, commDenv, commdLog - CommunicationsDaemon for the SIDLM cluster

DESCRIPTION

The Communications Daemon's task is to control operation in case of a failure in the cluster. In order to perform this task, there is a commD (uppercase "D" at the end) daemon running on each cluster node and a commd daemon running on the Cluster Console.

When a failure occurs (e.g. a node crashes), each commD reports its node's view of the problem to the Cluster Console's commd. commd decides which nodes will build up the new cluster and "panics" all other nodes as described below.

All commd activities are logged in a logfile in the commd directory and are shown in the logging window commdLog on the Cluster Console.

REQUIRED CONNECTIONS

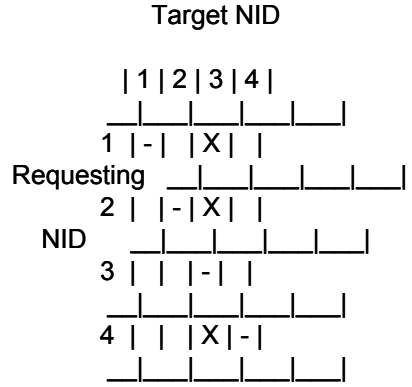
The Cluster Console commd exchanges messages with the nodes commDs via a separate network, which will be called the commd-LAN. The commd-LAN communication is based on UDP sockets with port number 7000. The port number can be changed in /etc/services (on the nodes as well as on the Cluster Console) in case of a collision with another service.

The cluster console needs to be connected to the node's console lines instead of real console terminals. The node's console I/O are mapped into X windows through Xscon(1M) .

FUNCTIONAL DESCRIPTION

When a node's DLM detects loss of heartbeat from any other node for a period longer than the network timeout (default 28 sec), it suspends all activities and reports this failure to its commD process. The commD creates a SHUTDOWN message, containing the information with which node(s) the communication broke down and sends it to the Cluster Console commd.

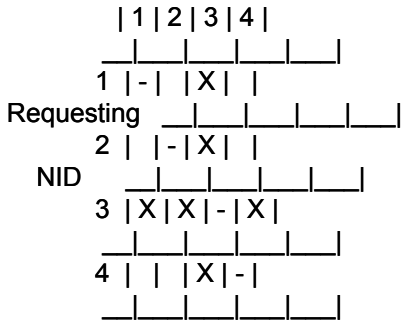
When commd receives a SHUTDOWN message from one node, it waits a certain time for the other nodes to report their view of the failure. commd inserts the failure information stemming from the SHUTDOWN messages into a matrix table. The resulting matrix is the base for further decisions. The following example shows the matrix for a 4 nodes cluster where node 3 failed.



Nodes 1, 2 and 4 reported that they could not communicate with node 3. Node 3 did not report anything.

A typical example for a Virtual Ethernet failure could look like this:

Target NID



Nodes 1,2 and 4 reported that they could not communicate with node 3. Node 3 reported that it could not communicate with all other nodes. Obviously, node 3 is disconnected from the Virtual Ethernet.

`commd` distinguishes two types of failure in the cluster:

1) Failure of one or more nodes
2) Failure in the Virtual Ethernet communication between living nodes

ERROR HANDLING

Node failure

In error situations of type 1) such that one or more nodes failed, `commd` has to ensure that a failed node will not resume its activities before rebooting. `commd` first checks if the failed node is in the "power off" state by checking the RS232 signals on the node's console line. If the RS232 lines are still set, `commd` forces the node to `ikdb` and issues the `ikdb` commands "pa" and "ex" on the console line which causes the node to quit `ikdb`, perform a PANIC and reboot immediately.

Virtual Ethernet Failure

When a Virtual Ethernet failure occurs, the situation is more complicated. There is an imminent danger that the cluster will be split into disjoint cluster segments, each containing one or more nodes. After the `commd` intervention, only one of these segments may remain as the new cluster. All nodes not contained in the remaining cluster have to be eliminated through `commd`. `commd` forces such a node to `ikdb` and performs the "pa" command. The node stays in `ikdb` and the system administrator has to fix the Virtual Ethernet problem before the node rejoins the cluster after its rebooting. For the decision which one of the disjoint segments is to be selected, commd uses two rules:

1) After the intervention, the cluster shall be as large as possible.
2) Nodes with a lower node ID have a higher priority than those with a higher node ID.

With the second rule, the cluster administrator can prioritize the cluster nodes during configuration (in `/etc/clustertab`).

After having eliminated all nodes that will no longer be part of the cluster, `commd` sends acknowledgement messages to the remaining nodes in the cluster. This causes them to perform a rebuild and resume activities in the remaining cluster.

REBUILD OF A CLUSTER

After a `commd` recovery action as explained in the previous section, or after a "normal" cluster rebuild due to nodes joining the configuration, the DLM and Oracle on the nodes cooperate to have a cluster running on a common database. One of the nodes acts in this context as a master, and the others act as slaves. At the end of this reconfiguration, the DLM locks held by the failed node at the time of its failure have been distributed to the remaining nodes. During the recovery procedure, DLM writes the following messages to its console:

```
rebuild starting due to <cause>
rebuild complete
cluster size <n>
```

where <n> is the number of remaining nodes and <cause> can be "node failure" after a recovery action of the `commd` or "reestablished connection" in case a new node joins the configuration.

REPAIR PROCEDURES

Node failures due to a crash or malfunctioning of a node

A malfunction in this context is any error on a node leading to a temporary or permanent hanging state. commd takes actions described in the previous section such that the cluster remains active, possibly with the elimination of one or more nodes. In this case, a dump of the failing nodes is taken. Please send the dump to SNI's customer support department. If the error was due to a hardware component and it can be identified, it should be replaced and the system rebooted.

Virtual Ethernet errors

commd takes actions described in the previous section such that the cluster remains active. All nodes not contained in the remaining cluster have to be eliminated by the commd. commd forces such a node to ikdb and performs the "pa" command. The node stays in ikdb. Either the system's administrator can figure out the reason for the error, e.g.

- failure of all Ethernet Controllers of the eliminated nodes
- Ethernet cabling problems

and repair the failure before rebooting the node, or can reboot the machine in single user mode and try to locate the cause of the error in this run level. If this does not help, the system administrator can prevent the machine from executing the script /etc/init.d/SIdlm (by moving it) and bring the machine in multiuser mode for further debugging.

ENVIRONMENT VARIABLES

The file /opt/SIdlm/commd/commdenv is created during installation of the SIcommd package on the Cluster Console. This file contains statements to set up commd variables which should be modified as required for the specific cluster configuration.

DISPLAY=`uname -n`:0
: This environment variable sets the correct X Window display for the log window. For an RM200 graphics console, the predefined value is suitable. It must only be changed if another X terminal device is used for output.

COMMD_HOST_NAME=`uname -n`
: If a private LAN is used for the commd communication (recommended configuration), replace the expression `uname -n` by the hostname of the Cluster Console in the commd's LAN. If the standard LAN is used (not recommended), then this setting can be left unchanged.

  Example: COMMD_HOST_NAME="cl_cons"

COMMD_DUMP=YES/NO
: This variable determines whether a node will perform a dump after leaving ikdb. If it is set to YES (default), then a dump will be taken. Otherwise, if set to NO, taking the dump will be omitted.

  Default is YES.

COMMD_MAX_TIME_TO_BOOT="time_interval_in_seconds"
: If the commd panics a node by issuing the ikdb command, under rare conditions, it could happen, that the node hangs after having exited ikdb. To guarantee an automatic reboot in this situation, commd inspects the node's console output. If commd does not detect the expression "Autoboot: Waiting" within the next COMMD_MAX_TIME_TO_BOOT seconds, the node will be reset.

  Default is 900.

COMMD_QUIT_KDB=YES/NO
: This variable determines, if a "panicked" node will leave ikdb immediately and will subsequently reboot automatically (default) or if the node will stay in ikdb. Independent of the setting of this variable, ikdb will not be quitted when a node was panicked by commd due to a Virtual Ethernet failure.

  Default is YES.

COMMD_MAX_DBG="commdDebug_file_size_in_bytes"

If commd has been started with the –d option, a file commdDebug will be created. If this file exceeds the COMMD_MAX_DBG size, then it will be renamed to commdDebug.old and a new commdDebug file will be created to log subsequent debug output.

Default is 50000.

COMMD_CRESET=YES/NO

If commd has decided which nodes will build the new cluster, it will "panic" all other nodes. This is usually done with help of the node's ikdb. If COMMD_CRESET=YES, commd will not use the ikdb. It will instead use another method to power off and reboot the ill nodes. On machines having a BDM (board debug monitor), commd uses either the creset or wreset BDM command. creset is used if COMMD_DUMP=NO or COMMD_WRESET=NO. We strongly recommend using the ikdb feature. If ikdb is not used, the node goes down without any file system sync, which could cause file system corruption on local hard drives.

Default is NO.

COMMD_WRESET=YES/NO

This environment variable is only used by commd for nodes having the BDM (board debug monitor). If the ikdb command fails on the node and COMMD_WRESET=YES, commd attempts to reboot the node by using the BDM wreset command. This command instructs BDM to produce a dump. Before doing this, BDM asks the operator to specify the dump device. To avoid this operator interaction, the user can specify COMMD_WRESET=NO. In this case, commd issues the creset command. However, one should be aware that the advantage of a fast reboot comes at the price of losing diagnostic information.

Irrespective of the setting of this variable, wreset will be used when a node was panicked by commd due to a Virtual Ethernet failure.

Default is YES.

After the installation of the SIdlm package on the cluster nodes, the file /opt/SIdlm/commD/commDenv contains a few statements to set up commD environment variables. Two of these environment variables must be set appropriately.

COMMD_HOST_NAME=`uname -n`

It depends on the Ethernet connection used, whether this setting is suitable or not. If the usual LAN is also used as the commd LAN, then this setting can be left unchanged. If there is a private LAN used for the commd communication, the expression `uname -n` must be replaced by the hostname of the cluster node in this LAN.

Example: COMMD_HOST_NAME="node2"

COMMD_SCON_NAME="cluster-console"

The expression "cluster-console" must be replaced by the hostname of the cluster console in the commd-LAN.

Example: COMMD_SCON_NAME="cl_cons"

FAIL SCRIPT

Optionally, the cluster administrator can create the shell script /opt/SIdlm/commD/fail_script on the cluster nodes. If it exists, it will be executed when the node commD detects that the Cluster Console commd failed (heartbeat monitoring). The script could be used for instance to send an e-mail to the cluster administrator. The contents of /opt/SIdlm/commD/fail_script could look like this:

```
mailx -s "commd failure" admin@mch.sni.de << EOF
Node 2: Cluster Console commd - heartbeat timeout
Administrator intervention required
EOF
```

COMMDLOG MESSAGES

All commd activities are logged in the file /opt/SIdlm/commd/commdLog and are shown in the commdLog window on the Cluster Console. Most of the messages are self explanatory, therefore only the error messages will be described in the following section.

COMMDLOG ERROR MESSAGES

`Bind call failed: kill old commd`
: The port number used for the `commd` communication is already in use. Most likely the `commd` process is already running.

`Nid <nid>'s commD has incompatible protocol version - reject`
: The node with node ID *nid* runs a `commD` with a message format which is incompatible to that of the Cluster Console `commd`. You should copy the `SIcommd` package from the node with node ID *nid* (`/opt/SIdlm/SIcommd`) to the Cluster Console and install it there.

`New node (NID <newnid>) tries to create a disjoint cluster:`
`Active cluster ( <nid> ... ) remains. New node stopped (ikdb)`
: `commd` detected an inconsistency. Node *newnid* has no LLT contact to the current cluster and therefore tries to create a disjoint cluster segment. Because this would destroy the database, `commd` eliminates node *newnid*. The Virtual Ethernet connections between *newnid* and the nodes contained in the current cluster have to be repaired, before *newnid* is allowed to be rebooted.

`New node (NID <newnid>) tries to create a disjoint cluster`
`Heartbeat timeout in active cluster ( <nid> ... ) - cleanup`
: Node *newnid* tries to register a disjoint cluster segment, but the nodes in the current cluster seem to be shut down, without having informed `commd`. `commd` discards its current view of the cluster and requests new notifications from all nodes. This message may be ignored.

`Wrong Cluster in NID <newnid>'s NOTIFY msg`
`Node <newnid> sees NIDs ' <nid> ... ',  Other nodes see NIDs '<nid> ...'`
: There is a mismatch in the node *newnid*'s view of the cluster. The cluster could have changed during the current registration phase. `commd` requests new notifications from all nodes. If the problem remains, i.e. the nodes still have different views of the cluster, there must be a problem in the Virtual Ethernet connection between the nodes.

`NID <nid> has incompatible Heartbeat value <sec> (previous <old>)`
: Node *nid*'s heartbeat value (seconds between heartbeats) differs from the value set in the other cluster nodes. The heartbeat value can be set with the command `dlmconfig -H` *sec* on the cluster nodes and should be the same on all nodes.

`Last cluster node (<nid>) left - cleanup`
: There was only one node in the cluster and this node is being shut down. There is no working cluster now. `commd` begins a new registration phase.

`commd is still in registration phase - ignore msg`
: The cluster console `commd` received a SHUTDOWN message from a node, but is not able to handle it, because it is still in the registration phase.

`SHUTDOWN msg received from unregistered NID <nid> (<node>) - ignore`
: A node that is not a member in the current cluster sent a SHUTDOWN message.

`WARN: Target Node not part of the cluster - reply ACK`
: A cluster node tried to eliminate another node which is not part of the current cluster. This can happen under rare circumstances, and it may be ignored.

FILES

On the Cluster Console:

`/opt/SIdlm/commd/commd`

`/opt/SIdlm/commd/commdenv`

`/opt/SIdlm/commd/commdLog`

On the Cluster Nodes:

`/opt/SIdlm/commD/commD`

`/opt/SIdlm/commD/commDenv`

`/opt/SIdlm/commD/fail_script`

SEE ALSO

cu(1), Xscon(1M), xscon(1M), dlm(7).

Fujitsu Siemens Computers Add-on Reference Manual

dlm(7)
SIDLM

NAME

dlm - introduction to distributed lock management

DESCRIPTION

Distributed lock management is a mechanism for synchronizing shared multiple resources by multiple processes. An application can use dlm to assist in the coordination of references and modifications to a resource, thereby maintaining the resource in a consistent state. The following section describes the terms and concepts used in relation to the dlm.

TERMS

domain
: Multiple name spaces (domains) allow nonconflicting use of dlm. The resources are configured per domain, with a usage limitation of the maximum number of clients. The domain is determined by the minor device number used to open the dlm.

resource
: A resource is any object that is accessed by an application. Some examples of resources are files, shared memory regions, and blocks in a file. Associated with each resource are value blocks, a queue of locks being granted access to the resource, and a queue of locks in the process of being granted access to the resource.

value block
: Associated with each resource is a value block that can be used to store shared information about the resource. In order to write to the value block, a process must hold the resource in Protected Write (PW) or Exclusive (EX) mode and must be converting the lock to the same or lower mode. (See access mode section below.) However, the value block can be read on any conversion request, and the value block is copied from the resource and returned to the requesting process when the request is granted. A value block can be in one of the following three states:

  LK_VS_NOVALUE
  : The resource is first created and marked as uninitialized.

  LK_VS_VALUE
  : A process writes a value to the value block.

  LK_VS_DUBVALUE
  : A process that holds a lock on the resource in Protected Write (PW) or Exclusive (EX) mode has crashed. The value block of the resource is then marked as dubious. A value block in a dubious state contains information that is invalid.

lock A lock is used to obtain rights to a resource. The rights to a resource are based on the access modes, which are also referred to as lock levels. The following access modes are valid:

  LK_NL Null mode. Initially, all locks are created in this mode. This mode is considered the lowest lock level.

  LK_CR Concurrent Read mode. A process holding a lock in this mode can read in an unprotected environment; other processes can read and write to the associated resource.

  LK_CW Concurrent Write mode. A process holding a lock in this mode can read and write in an unprotected environment; other processes can read and write to the associated resource.

  LK_PR Protected Read mode. This is a shared lock. When a lock is held in this mode, no process can write to the associated resource; however, multiple reads are allowed.

  LK_PW Protected Write mode. Only one process at a time can hold a lock at this level. That process is allowed to modify the associated resource, while other processes can only perform unprotected reads.

`LK_EX` Exclusive mode. A process holding a lock in this mode is given exclusive access to a resource. All other processes are denied read or write access to the resource. This mode is considered the highest lock level.

A process may request that the lock mode for a resource be changed. A conversion request succeeds immediately if the conversion is from one lock level to an equal or lower level. This is called *down conversion*. If a conversion request is from one lock level to a higher level, the request is not granted immediately if there is a conflicting lock mode in force, or if there is another conversion request already in the convert queue. This is called *up conversion*.

The following table describes the access modes that can be granted on a resource when the same resource is held by other processes:

Type of Lock	Mode of Currently Granted Locks					
Request	NL	CR	CW	PR	PW	EX
NL	Grant	Grant	Grant	Grant	Grant	Grant
CR	Grant	Grant	Grant	Grant	Grant	Queue
CW	Grant	Grant	Grant	Queue	Queue	Queue
PR	Grant	Grant	Queue	Grant	Queue	Queue
PW	Grant	Grant	Queue	Queue	Queue	Queue
EX	Grant	Queue	Queue	Queue	Queue	Queue

`blocking asynchronous trap`
This term, known as BAST, is a message that notifies the process that holds a lock on a resource that another process is requesting a lock in a conflicting mode on the same resource. This message causes a user-specified callback routine to be invoked which forces the blocking process to convert the lock to a lower mode so that the waiting process can obtain the requested lock.

`deadlock`
A deadlock occurs when two processes each hold a different resource in Exclusive mode, and each process attempts to obtain the resource held by the other process. If neither process is willing to give up its access rights, then a deadlock occurs. Whenever a request is made to convert a lock and the request cannot be granted immediately, `dlm` performs deadlock detection. If the conversion request will cause a deadlock, the conversion request fails.

`groups`
The concept of group allows any processes in a group to manipulate locks held by any member of the group. One process may create a group, and other processes can attach to that group. A process may detach from a group. Locks held by the members of a group are not given up until all processes attached to the group have terminated or until the process that created the group terminates.

`transaction identifier`
A concurrent environment is provided through the use of Transaction Identifiers (XIDs). This allows two threads of control working on behalf of a single transaction to each acquire and hold an Exclusive mode lock concurrently. Such a task can be accomplished by providing routines to specify transaction identifiers when opening a lock.

EXAMPLES

Assume there is a resource called `rsrc`, and two processes each hold a Null mode lock on `rsrc`. The first process converts its lock to Protected Read mode. There is no conflict, and the request to convert lock modes is granted immediately. Resource `rsrc` may not be written by any other processes, but multiple reads are allowed. If the second process wants to convert to Exclusive mode, the request is queued since it conflicts with the request of the first process. When the first process converts back to a Null lock mode, then the second process is allowed exclusive access to resource `rsrc`.

SEE ALSO

clustertab(4), dlmconfig(8).

Parallel Server System Administrator's Guide.

Fujitsu Siemens Computers Add-on Reference Manual

dlm(7)
SPS

NAME

dlm - introduction to distributed lock management

DESCRIPTION

Distributed lock management is a mechanism for synchronizing shared multiple resources by multiple processes. An application can use dlm to assist in the coordination of references and modifications to a resource, thereby maintaining the resource in a consistent state. The following section describes the terms and concepts used in relation to the dlm.

TERMS

domain
: Multiple name spaces (domains) allow nonconflicting use of dlm. The resources are configured per domain, with a usage limitation of the maximum number of clients. The domain is determined by the minor device number used to open the dlm.

resource
: A resource is any object that is accessed by an application. Some examples of resources are files, shared memory regions, and blocks in a file. Associated with each resource are value blocks, a queue of locks being granted access to the resource, and a queue of locks in the process of being granted access to the resource.

value block
: Associated with each resource is a value block that can be used to store shared information about the resource. In order to write to the value block, a process must hold the resource in Protected Write (PW) or Exclusive (EX) mode and must be converting the lock to the same or lower mode. (See access mode section below.) However, the value block can be read on any conversion request, and the value block is copied from the resource and returned to the requesting process when the request is granted. A value block can be in one of the following three states:

  LK_VS_NOVALUE
  : The resource is first created and marked as uninitialized.

  LK_VS_VALUE
  : A process writes a value to the value block.

  LK_VS_DUBVALUE
  : A process that holds a lock on the resource in Protected Write (PW) or Exclusive (EX) mode has crashed. The value block of the resource is then marked as dubious. A value block in a dubious state contains information that is invalid.

lock A lock is used to obtain rights to a resource. The rights to a resource are based on the access modes, which are also referred to as lock levels. The following access modes are valid:

  LK_NL Null mode. Initially, all locks are created in this mode. This mode is considered the lowest lock level.

  LK_CR Concurrent Read mode. A process holding a lock in this mode can read in an unprotected environment; other processes can read and write to the associated resource.

  LK_CW Concurrent Write mode. A process holding a lock in this mode can read and write in an unprotected environment; other processes can read and write to the associated resource.

  LK_PR Protected Read mode. This is a shared lock. When a lock is held in this mode, no process can write to the associated resource; however, multiple reads are allowed.

  LK_PW Protected Write mode. Only one process at a time can hold a lock at this level. That process is allowed to modify the associated resource, while other processes can only perform unprotected reads.

`LK_EX` Exclusive mode. A process holding a lock in this mode is given exclusive access to a resource. All other processes are denied read or write access to the resource. This mode is considered the highest lock level.

A process may request that the lock mode for a resource be changed. A conversion request succeeds immediately if the conversion is from one lock level to an equal or lower level. This is called *down conversion*. If a conversion request is from one lock level to a higher level, the request is not granted immediately if there is a conflicting lock mode in force, or if there is another conversion request already in the convert queue. This is called *up conversion*.

The following table describes the access modes that can be granted on a resource when the same resource is held by other processes:

Type of Lock	Mode of Currently Granted Locks					
Request	NL	CR	CW	PR	PW	EX
NL	Grant	Grant	Grant	Grant	Grant	Grant
CR	Grant	Grant	Grant	Grant	Grant	Queue
CW	Grant	Grant	Grant	Queue	Queue	Queue
PR	Grant	Grant	Queue	Grant	Queue	Queue
PW	Grant	Grant	Queue	Queue	Queue	Queue
EX	Grant	Queue	Queue	Queue	Queue	Queue

`blocking asynchronous trap`
This term, known as BAST, is a message that notifies the process that holds a lock on a resource that another process is requesting a lock in a conflicting mode on the same resource. This message causes a user-specified callback routine to be invoked which forces the blocking process to convert the lock to a lower mode so that the waiting process can obtain the requested lock.

`deadlock`
A deadlock occurs when two processes each hold a different resource in Exclusive mode, and each process attempts to obtain the resource held by the other process. If neither process is willing to give up its access rights, then a deadlock occurs. Whenever a request is made to convert a lock and the request cannot be granted immediately, `dlm` performs deadlock detection. If the conversion request will cause a deadlock, the conversion request fails.

`groups`
The concept of group allows any processes in a group to manipulate locks held by any member of the group. One process may create a group, and other processes can attach to that group. A process may detach from a group. Locks held by the members of a group are not given up until all processes attached to the group have terminated or until the process that created the group terminates.

`transaction identifier`
A concurrent environment is provided through the use of Transaction Identifiers (XIDs). This allows two threads of control working on behalf of a single transaction to each acquire and hold an Exclusive mode lock concurrently. Such a task can be accomplished by providing routines to specify transaction identifiers when opening a lock.

EXAMPLES

Assume there is a resource called `rsrc`, and two processes each hold a Null mode lock on `rsrc`. The first process converts its lock to Protected Read mode. There is no conflict, and the request to convert lock modes is granted immediately. Resource `rsrc` may not be written by any other processes, but multiple reads are allowed. If the second process wants to convert to Exclusive mode, the request is queued since it conflicts with the request of the first process. When the first process converts back to a Null lock mode, then the second process is allowed exclusive access to resource `rsrc`.

SEE ALSO

`dlmconfig`(8) .

Parallel Server System Administrator's Guide.

Fujitsu Siemens Computers Add-on Reference Manual

dptg2(7)

NAME

dptg2 - terminal multiplexer protocol

DESCRIPTION

The DPTG2 protocol is an SNI-specific terminal protocol for using BA47 and BA80 (9766) screens. Operation of PCs with DPTG2 protocol emulation is also supported. Up to 16 virtual channels can be used on an asynchronous V.24 line with this protocol.

Each of the virtual channels is available to the user as an independent terminal in /dev/term. However, it is not possible to set the physical line parameters with an ioctl system call. A flow control is allowed for each channel, which prevents a buffer overflow in the screen. Apart from the restriction regarding the physical parameters of the line, a complete functionality of UNIX system V Release 4 is offered.

The components of DPTG2 are contained in the Reliant UNIX package SIdptg2. This pakkage is mounted in the directory /opt/dptg2.

The DPTG2 protocol offers the following features:

- The DPTG2 protocol allows the connection and startup of DPTG2 terminals. Operation of the PROM software for command entry (ASCII characters > 0x20 and control characters CR, LF) is supported. Operation of VT terminals with the DPTG2 protocol is not supported. If these terminals are used on lines configured for DPTG2 terminals, it may be necessary to generate a new operating system.
- The control characters XON and XOFF are used only for the flow control of the physical line.
- If the terminal does not support the DPTG2 protocol (e.g. because a workstation program has not yet been loaded), the channels respond to the system calls in the same way that they would if the terminal was switched off.
- The setup procedure for the terminals is transparent to the user. The channels are only cleared when the terminal has been successfully set up and when it is ready for the DPTG2 protocol, thus as if the terminal was already switched on. None of the channels is used for the setup (as was the case with the previous protocol version, DPTG1).
- The DPTG2 protocol can be used for asynchronous V.24 lines as well as for TCP/IP network connections. This means that the screens can be connected either directly to the Reliant UNIX system, or remotely to a terminal server or DOS-PC with DPTG2 emulation.

The DPTG2 protocol is defined for an asynchronous connection between the workstation program in the terminal and a STREAMS multiplexer in the Reliant UNIX host. On the host side, the DPTG2 multiplexer is connected with a TTY driver or a TCP/IP socket (using the system call ioctl(I_LINK)) after the terminal has been recognized as being available. The DPTG2 connection between host and screen is set up by a monitor as part of the SAF (Service Access Facility). This monitor is called muxmon and is part of the DPTG2 package.

The DPTG2 monitor muxmon can be managed with the muxadm(1M) command, which is included in the DPTG2 package. It supports the special features of the DPTG2 monitor. muxadm is used by the termadd(1M) command for the configuration of a DPTG2 channel. Commands are forwarded to sac(1M) during configuration, whereby the channel is incorporated in the DPTG2-specific administration. A monitor incarnation can be created and started by termadd(1M) if necessary.

The termadd(1M) command is not included in the DPTG2 package, as it can also be used to configure other types of screen.

The muxmon(1M) monitor not only sets up a DPTG2 connection to the screen, but also loads the screen if necessary. It also monitors the status of the screen. If the monitor is switched off (or the network connection crashes), the I_LINK link between the multiplexer and driver (or socket in the case of remote connections) is removed. The DPTG2 channel responds to system calls in the same way that it would if the terminal were switched off.

A new feature of DPTG2 is that, unlike DPTG1, the terminal setup for an application is transparent, and the `CLOCAL` functionality is available on a channel-specific basis.

DIAGNOSTICS

The DPTG2 components generate status and error messages for Logging V3.0 for diagnostic purposes. The component number for messages in the area of DPTG2 is 36 (`SIdptg2`).

The error numbers for the individual components of the DPTG2 system are:

Label	Name	Error number
Multiplexer	`dptg2mux`	1
Link monitor	`muxmon`	6
Admin command	`muxadm`	7

NOTES

The workstation programs, which can be loaded onto the BA80, are not included in the `SIdptg2` package. By default, they are under `/opt/xb`. If they are not available, the `SIxb` package must be installed.

The `SIdptg2` package dos not have any commands for the configuration of DPTG channels. These commands can be found in the `SIterms` package.

FILES

`/opt/dptg2`
`/opt/xb`

SEE ALSO

`muxadm`(1M), `muxmon`(1M), `sac`(1M), `termadd`(1M), `ioctl`(2).

Fujitsu Siemens Computers Add-on Reference Manual

draid(7)

NAME

`draid` - system driver for the two channels of a RAID box

DESCRIPTION

`draid` is a driver in the operating system kernel and is located between the general I/O systems and the physical RAID driver.

The `draid` driver is connected to the general I/O systems over a device node, and uses two or more physical RAID device nodes. The `draid` driver transmits the I/O systems' requirements to a physical RAID device node. If one of the device nodes fails for any reason, the `draid` driver transmits all subsequent I/O requirements over an alternative physical RAID device node which is still available. The tasks intended for the failed device node are repeated over the device node that is still available.

SEE ALSO

`draidd`(1M), `draid`(4).

Fujitsu Siemens Computers Add-on Reference Manual

ep(7)

NAME

`ep` - event profiler

DESCRIPTION

A problem that often arises when writing a program is that a part of the program allocates resources, such as memory areas, and then either does not free these resources at all or frees them several times.

Within the system kernel, the event profiler is the component that allows the developer to collect information on such resources. These resources are termed *OBJECTS* below. Examples of *OBJECTS* include memory areas, locks, file descriptors or other items that can be identified by a unique key.

Various events can occur during the service life of an *OBJECT*. Standard events include, for example, creating and freeing an *OBJECT*.

An example of an *OBJECT* is a streams message block (msgb). This type of *OBJECT* is

- created in the `allocb()` or `dupb()` functions,
- processed in the `putq()`, `getq()`, `putbq()` and `putnext()` functions and
- freed again in the `freeb()` function.

The forwarding of a message to another streams module by means of a `putnext()` can be very interesting in the case of the debugger and should therefore also be logged.

The event profiler must be informed of the creation, use and freeing of the *OBJECTS*. There are three different types of events here:

- Following creation of a new *OBJECT*S, the event profiler has to be informed that information should now be collected for this *OBJECT*. The creation event is stored automatically in the process by the event profiler for the *OBJECT*. The event profiler stores the ID of the trace point, or the trace ID, a time stamp and the stack backtrace of the caller, if no trace-specific data was handed over when the function was called. In this case, it is possible to store the values of certain variables in addition to the stack backtrace of the caller. Based on this information, it is very easy to determine who created the *OBJECT* and when.
- Other events can also be stored in the profile of the event profiler. These events are control points at which the system programmer can log when and where an *OBJECT* changed its status or its position. As is the case when the *OBJECT* is created, trace-specific information can also be stored for these events instead of the stack backtrace of the caller.
- The event profiler has to be informed if the *OBJECT* is freed again (release event). The behavior of the event profiler in the case of this event depends on its mode. This is described below.

The event profiler can be started in two different modes:

- The history of all *OBJECTS* created is logged in `PROFILING` mode. When an *OBJECT* is freed, all information stored in the event profiler for this *OBJECT* is also freed. The time stamps stored by the event profiler for each event can be used by the system programmers to find the memory leak. If an *OBJECT* was created an hour previously, for example, this is a possible candidate.
- Multiple release can be recognized in `MFREE` mode. If an *OBJECT* is freed, not all information for this *OBJECT* is freed. Instead, a release event is stored with the stack backtrace of the caller. If the *OBJECT* is freed again, the event profiler can detect this using this event. A PANIC is normally initiated in this case. The system programmer can find the first freeing entity based on the stack backtrace of the caller stored in the release event. The second freeing entity can be established based on the stack backtrace of the caller of the process which initiated the PANIC.

The position in the code at which the event profiler is notified of an event, by one of the `ep` event functions (see below) being invoked, is termed a trace point.

The trace points are all given unique numbers to help the developer to determine which events are of interest in a particular situation; these trace points are termed trace IDs. The numbers comprise bit masks in which

only a single bit is set. By setting the respective bits, a global bit mask – the so-called trace mask – can then be used to determine which trace points are active. The `ep_tracemasks` trace mask [*number_of_CLIENT*] is set using `crash`(1M) when the event profiler is initialized in accordance with the mode.

In order to be able to examine several parts of the kernel at the same time, the event profiler differentiates between so-called *CLIENTS*. A *CLIENT* is a part of the kernel in which trace points have been integrated for the event profiler. If, for example, an error were to be found in the kernel space management, a new *CLIENT* would have to be defined (see below). Trace points then have to be incorporated in this part of the kernel. An example of a part of the kernel for which this has already taken place is `Streams`.

CLIENTS are identified by means of unique numbers [`0`,*number_of_CLIENTS*–1]. To create a new *CLIENT*, a `#define` statement has to be inserted in `sys/ep.h` which assigns such a number to the name of the new *CLIENT*. The `EP_CLIENT_LAST` `#define` statement has to be adapted so that it still specifies the last number used. The start value for the trace mask for this *CLIENT* (`ep_tracemasks` [*new_CLIENT number*]) must be set to `~0`. This means that all trace points are active.

The functions that can be used to inform the event profiler of the events, all have the following parameters in common:

traceid
: The *traceid* is a unique number per *CLIENT* for a specific trace point.

client
: *client* is a unique number for this *CLIENT*.

key
: *key* identifies the *OBJECT*. The address of a buffer can be used here, for example `struct msgb *bp`.

skipcallers
: *skipcallers* specifies the number of callers that can be skipped when storing the stack backtrace of the caller. This can be very useful if the name of the last caller contains no further useful information. The value `0` means that the storage of the stack backtrace of the caller begins with the caller of the `ep` function. However, since this is usually identified on the basis of the trace point number (trace ID), the value `1` is generally used.

The trace points should always remain integrated in the kernel. This is the only way to guarantee that the event profiler can be used as soon as an error occurs. However, to ensure that the trace points do not take an unnecessary amount of time when the event profiler is disabled, the `ep` function calls have been encapsulated in `#define` statements. These only invoke the functions when necessary. Only these macros should therefore be used (see `sys/ep.h`). The only difference in calling the macros and calling the functions is that the macros are specified in uppercase.

The `ep` functions are described below:

```
void ep_create(ulong_t traceid, int client, ulong_t key,
     ulong_t maxsize, addr_t buf, ulong_t len, int skipcallers)
```

The event profiler is informed that a new *OBJECT* has been created. The event profiler automatically stores the creation event.

maxsize
: Maximum number of bytes to be stored as trace-specific data by any `ep` function calls within this *CLIENT*. This number must be the same for all `ep_create()` calls within this *CLIENT*.

buf
: Address of trace-specific data or `NULL`.

len
: Length of trace-specific data for this event or 0. This length must of course always be less than *maxsize*.

```
void ep_destroy(ulong_t traceid, int client, ulong_t key, int skipcallers)
```

The event profiler is informed that the *OBJECT* has been freed again.

```
void ep_push(ulong_t traceid, int client, ulong_t key,
     addr_t buf, int len, int skipcallers)
```

A further event is stored in the event profiler.

buf
: Address of trace-specific data or `NULL`.

len Length of trace-specific data for this event or 0. This length must of course always be less than the *maxsize* used with `ep_create()`.

A further function is also made available in addition to these event functions:

`int ep_getcallers(int client, ulong_t *buf, int skipcallers)`

You can use this function to store the stack backtrace of the caller in a trace-specific data buffer. The function specifies the number of stored callers. If fewer callers could be stored than configured for this *CLIENT* using `crash`, the remainder is filled with `NULL`.

buf Address of a buffer in which the stack backtrace of the caller is to be stored.

CLIENT: Streams

`Streams` is an initial *CLIENT*, for which trace points have been integrated. The event profiler aims to detect when memory is being leaked or when `Streams` message blocks are freed twice. The following trace points have been integrated for this purpose:

Function	Trace ID	Event function
allocb	0x1	ep_create()
dupb	0x2	ep_create()
getq	0x4	ep_push()
getq	0x8	ep_push()
putq	0x10	ep_push()
putbq	0x20	ep_push()
putnext	0x40	ep_push()
freeb	0x80	ep_destroy()
freeb	0x100	ep_destroy()
freeb	0x200	ep_destroy()
freeb	0x400	ep_destroy()
freeb	0x800	ep_destroy()
freeb	0x1000	ep_destroy()

Since `freeb()` contains six different calls for `free_msgblock()` and `free_mdbblock()`, there are six different trace IDs for `freeb()` in the above list.

EXAMPLES

To identify a `Streams` message block, you need the address of the data header as well as the address of the message header. Consequently, `Streams` uses special macros which store the address of the data header in the trace-specific data buffers in addition to the stack backtrace of the caller:

```
#define EP_PROFILING□
#ifdef EP_PROFILING□
#include "sys/ep.h"□
#define ME      EP_CLIENT_STREAMS□
#define EP_STR_INIT_BUF(mp)\□
 ep_clnt_streams_t buf;\□
 int n = ep_getcallers(ME, &buf.stk[0], 1);\□
 buf.datab = (ulong_t)(((struct msgb *)(mp))->b_datap)□
#define EP_STR_NEW(mask, mp)\□
 if (ep_active[ME]) {\□
    EP_STR_INIT_BUF(mp);\□
    ep_create(mask, ME, (ulong_t)(mp), (n+1) * sizeof(ulong_t),\□
```

```
  (addr_t)&buf, (n+1) * sizeof(ulong_t),
0);\
        }
#define EP_STR_LOG(mask, mp)\
 if (ep_clients[ME]) {\
    EP_STR_INIT_BUF(mp);\
    ep_push(mask, ME, (ulong_t)(mp), (addr_t)&buf,\
  (n+1) * sizeof(ulong_t), 0);\
        }
#define EP_STR_DELETE(mask, mp)\
 if (ep_clients[ME])\
    ep_destroy(mask, ME, (ulong_t)mp, 1);
#endif
```

A trace point in `freeb()` would then look as follows, for example:

```
    ...
    EP_STR_DELETE(0x80, bp);
    free_msgblock(bp);
```

SEE ALSO

`crash`(1M) .

Fujitsu Siemens Computers Add-on Reference Manual

hdlcio(7)
RM400 only

NAME

`hdlcio` - HDLC-specific ioctl calls

SYNOPSIS

```
#include <hdlcio.h>
int ioctl(fildes, command, arg)
int fildes, command;
struct hdlcio *arg;
```

DESCRIPTION

The HDLC-specific `ioctl` commands are used for setting the line and protocol parameters and for reading status information. The synopsis of the system call is compatible with the default `ioctl(2)` . The parameters to be read or set are created in a structure of the type `struct hdlcio`, the address of which is transferred as the argument `arg`.

The following commands are supported:

- `HDLCRSET`
- `HDLCSET`
- `HDLCGET`
- `HDLCWAIT`
- `HDLCINQGET`

The HDLC parameters relating to a device are set using the `HDLCSET` and `HDLCRSET` commands. These commands are identical, and are only supported for reasons of compatibility.

The HDLC parameters relating to a device can be read using the `HDLCGET` command.

With the `HDLCINQGET` command, the parameters are copied to the `hdlcio` structure specified by `arg`, and data input from the terminal is stopped. The amount of data in the input queue is reported. Since the input for this device has been blocked, this amount remains unchanged until the data is retrieved or the device is closed. If all of the data was retrieved from the application, the input for this device is cleared again, and data can be received once more.

The `HDLCWAIT` command is used to check an application to determine whether an HDLC terminal is responding to polls from the primary host. If this is the case, the command is canceled immediately. Otherwise, the procedure is blocked until the terminal responds.

The individual parameters are defined in the structure `hdlcio`.

```
struct hdlcio {
       struct cb_count    hdlcio_count;    /* HDLC counter        */
       struct cb_l_timer  hdlcio_l_timer;  /* HDLC line timer     */
       struct cb_t_timer  hdlcio_t_timer;  /* HDLC terminal timer */
       struct cb_output   hdlcio_output;   /* HDLC output flags   */
       struct cb_input    hdlcio_input;    /* HDLC input flags    */
};
```

The structures used in the `hdlcio` structure are defined as follows:

```
struct cb_count {          /* HDLC-spec. counter                 */
       ushort ActPCnt;     /* Retry counter for fast polls       */
       byte   RptCnt;      /* Retry counter for I frames         */
       byte   RespTOCnt;   /* Response timeout counter           */
};
```

`ACTPCnt`

Active Poll Count (retry counter for fast polls). Specifies how many poll attempts are to be made to

send or receive an I frame without delay. The counter is reset every time an I frame is sent or received.

When the timer expires, the primary host for this terminal switches to "inactive" mode.

Default: 2350 (this corresponds to a period of 2.5 minutes with a polling frequency of 50 ms).

`RptCnt`

Repeat Count (retry counter for I frames). Specifies how often an unacknowledged I frame is repeated, provided the terminal has responded.

When this counter expires, an error message is generated and the terminal switches to "disconnected mode".

`RespTOCnt`

Response Timeout Count (response timeout counter). Specifies how often a job is repeated if the terminal does not respond. The parameter is no longer evaluated. The `RptCnt` counter is used.

Default: 6.

```
struct cb_l_timer {          /* Line-spec. timer        */
       ushort   M1Timer;     /* M1 timer in msec.       */
       ushort   M2Timer;     /* M2 timer in msec.       */
       ushort   CharTimer;   /* Charactertimer in msec. */
       ushort   LineType;    /* Line type               */
       ushort   ClockMode;   /* Clock mode              */
       ushort   TrailPads;   /* Delayed S2 return       */
};
```

`M1Timer`

`M1` monitoring timer. Specifies the maximum waiting time `M1` (in msec.).

This parameter is no longer evaluated, but is supported for compatibility reasons.

`M2Timer`

`M2` monitoring timer. Specifies the maximum waiting time for the `M2` message.

If `M2` was not set within this time, the frame is not sent and the terminal switches to error status.

Default: 3000 ms.

`CharTimer`

Character Timer. Specifies the maximum time for sending and receiving the entire frame. This means that the process of sending and receiving the entire frame is monitored. The character timer depends on the line speed and the length of the frame. The timer should be set to cover all speeds. If a frame cannot be sent or received within the specified time, the terminal switches to error status.

Default: 5000 ms.

`LineType`

Line type. Specifies whether the line is a four-wire or two-wire line. If it is a two-wire line, the `S2` signal must be canceled after every frame is sent; this does not apply to a four-wire line.

For a two-wire line, set `LineType` to `TWOWIRE`.

For a four-wire line, set `LineType` to `FOURWIRE`.

This parameter only refers to the physical operation of the line. Logically, half-duplex mode is always used.

Default: `FOURWIRE`.

`ClockMode`

Clock mode. This parameter is no longer evaluated, but is supported for compatibility reasons. The clock mode is set by selecting the speed.

Default: 0

`TrailPads`

Delayed `S2` return. This parameter is not supported.

```
struct cb_t_timer {             /* Terminal-spec. timer          */
```

```
        ushort   SnrmRespTimer;  /* Response timer in DM in msec. */
        ushort   RespTimer;      /* Response timer in NRM in msec.*/
        ushort   M5Timer;        /* M5 timer in msec.             */
        ushort   IActPoll;       /* Poll spacing for              */
                                 /* fast polls in msec.           */
        ushort   IInActPoll;     /* Poll spacing for              */
                                 /* delayed polls in msec.        */
        ushort   INoRspPoll;     /* Poll spacing in               */
                                 /* no-response mode in msec.     */
        ushort   IStdbyPoll;     /* Poll spacing in               */
                                 /* standby-poll mode in msec.    */
};
```

`SnrmRespTimer`
: Set Normal Response Mode Timer. Specifies the maximum waiting time for a "quick" response (e.□g. UA). The job is repeated after the timer has expired.

  Default: 32 ms.

`RespTimer`
: Response timer in normal response mode (NRM). Specifies the maximum waiting time for a response in NRM, if an I frame is expected. The last job is repeated after the timer expires.

  Default: 300 ms.

`M5Timer`
: `M5` monitoring timer. Specifies the maximum time the last secondary polled will wait before canceling the `S2` signal. After the timer has expired, the next secondary is polled. If `S2` is not canceled after the specified time hs elapsed, the next secondary is nonetheless polled.

  Default: 10 ms.

`IActPoll`
: Active Poll. Polling frequency for fast polls. Specifies the frequency with which a terminal in active mode is to be polled.

  Default: 50 ms.

`IINActPoll`
: Inactive Poll. Polling frequency for delayed polls. Specifies the frequency with which a terminal in inactive mode is to be polled.

  A terminal is said to be inactive if the retry counter for `ACTPCnt` has expired.

  Default: 500 ms.

`INoRspPoll`
: No Response Poll. Polling frequency for terminals in no response mode. Specifies the frequency with which a terminal in no response mode is to be polled.

  A terminal is said to be in no response mode if a long period of time has elapsed since it answered a poll. The terminal is probably switched off or is not connected.

  Default: 10000 ms.

`IStdbyPoll`
: Standby Poll. Polling frequency for terminals in standby poll mode. Specifies the frequency with which a terminal in standby poll mode is to be polled.

  A terminal is said to be in standby poll mode if an `open` exists but the terminal has not yet answered.

  Default: 5000 ms.

```
struct cb_output {          /* Parameter for output control    */
        byte    oflag;      /* Flag for setting                */
                            /* blocked write                   */
        ushort  SUsrDataL;  /* Send data length                */
                            /* Default: 100 chars.             */
};
```

`oflag`
: output flag. Flag for setting the blocked `write`/`read`. The values `UNBLOCKED` and `BLOCKED` are permitted here. The default value is set if `UNBLOCKED` is used. The value `BLOCKED` activates blocked mode by assigning control of the data field of a HDLC frame to the application.

  Default: `UNBLOCKED`.

`SUsrDataL`
: Send User Data Length. The application can use this parameter to set the maximum length of a frame. A maximum length of 1 KByte is permitted. If a `write` command is issued with *n* data, this function can only be executed correctly if *n* is less than or equal to the maximum size of the send frame.

  If a value >1 KByte is set using `ioctl` for `SUsrDataL`, `ioctl` is rejected with the error `EINVAL`.

  Default: 100.

`InqAnz`
: Volume of data in the input queue. The application can use this field to poll the data available in the input queue. This field is only filled in using the `HDLCINQGET` command.

NOTES

The link-, or line-specific parameters can be set using any device that belongs to this link or line. They are set automatically for all devices operating on this link/line.

FILES

`/usr/include/sys/hdlcio.h`

SEE ALSO

`ioctl`(2), `open`(2), `read`(2), `write`(2), `hdlcdefs`(4), `hios`(7), `sih`(7).

Fujitsu Siemens Computers Add-on Reference Manual

hios(7)
RM400 only

NAME

`hios` - HDLC-ioctl for the reading in of statistical data

SYNOPSIS

```
#include <hios.h>
int ioctl(fildes, command, arg)
int fildes, command;
struct HdlcIOState *arg;
```

DESCRIPTION

The `ioctl` command `HIOS_GET` can be used to output statistical data.

The individual statistics parameters are defined in the structure `HdlcIOState`.

```
struct HdlcIOState {              /* HDLC status structure */
       unsigned int   HIO_Time    /* HDLC send-receive time (ms) */
       unsigned int   HIO_CONE    /* HDLC line connection error */
       unsigned int   HOS_IFrame  /* HDLC send Information frame */
       unsigned int   HOS_SFrame  /* HDLC send Supervisory frame */
       unsigned int   HOS_UFrame  /* HDLC send Unnumbered frame */
       unsigned int   HOS_Count   /* HDLC send count */
       unsigned int   HOS_Retry   /* HDLC send retry count */
       unsigned int   HIS_IFrame  /* HDLC receive Information frame */
       unsigned int   HIS_SFrame  /* HDLC receive Supervisory frame */
       unsigned int   HIS_UFrame  /* HDLC receive Unnumbered frame */
       unsigned int   HIS_Count   /* HDLC receive count */
       unsigned int   HIS_CRCE    /* HDLC receive CRC error */
       unsigned int   HIS_TimeE   /* HDLC receive timeout error */
       unsigned int   HIS_SizeE   /* HDLC receive size error */
};
```

`HIO_Time`
: Time in ms, during which the driver for the line (send, receive) was active.

`HIO_CONE`
: Counter to determine how often the line was started and the connection had to be set up again.

`HOS_IFrame`
: Number of I frames sent.

`HOS_SFrame`
: Number of S frames sent.

`HOS_UFrame`
: Number of U frames sent.

`HOS_Count`
: Number of data bytes sent.

`HOS_Retry`
: Number of retries.

`HIS_IFrame`
: Number of I frames received.

`HIS_SFrame`
: Number of S frames received.

`HIS_UFrame`
: Number of U frames received.

`HIS_Count`
: NUmber of data bytes received.

`HIS_CRCE`
: Number of CRC errors.

`HIS_TimeE`
: Number of timeout errors for a response.

`HIS_SizeE`
: Number of size errors.

NOTES

The counters are set to the link during the first `open(2)`, and reset to 0 on selection. In order to ensure accurate results, the statistical data should only be output by one application at a time.

FILES

`/usr/include/sys/io/hios.h`

SEE ALSO

`ioctl(2)`, `open(2)`, `hdlcio(7)`, `sih(7)`.

Fujitsu Siemens Computers Add-on Reference Manual

hs(7)

NAME

`hs` - High Sierra and ISO 9660 file system

DESCRIPTION

The `hs` file system is implemented using the virtual file system (VFS) of System V Release 4. This way the access of files on a CD-ROM is transparent. So all system calls for a read-only file system can be used [e.g. `open(2)`, `mount(2)`, `read(2)`, `mmap(2)` ...].

The High Sierra or ISO 9660 file system defines the following format for files and directories:

`NAME.EXT;VERSION`

`NAME` and `EXT` consist of upper case letters, digits and underscores "_". `VERSION` is a number.

FILE NAME MAPPING

UNIX file names are displayed without any name mapping; e.g. output of `ls(1)` might look like:

```
PICTURE.PS;3 PICTURE.PS;2 PICTURE.PS;1
```

A special option of the `mount(1M)` command turns on a file name mapping to provide compatibility for DOS applications on exported file systems. The following rules are implemented:

- All upper case letters are converted to lower case.
- When searching for a file name no distinction is made between upper and lower case letters:
- The version number of the name is truncated. If a file is accessed the highest version of it is used. This behaviour can be circumvented by adding the version number to the file name.

SEE ALSO

generic `mount(1M)`, `hs`-specific `mount(1M)`.

Information processing – Volume and file structure of CD-ROM for information interchange, Edition 1988-09-01, Ref.Nr ISO9660:1988(E)

Fujitsu Siemens Computers Add-on Reference Manual

msw(7)

NAME

`msw` - MAC Switch Driver (Virtual Ethernet Driver)

DESCRIPTION

The MAC Switch driver is a pseudo network driver that provides a single virtual network view of two or more physical network connections. If one of the underlying physical network channels fails, that physical network channel is marked as offline and the application network packets are automatically switched to other online physical network channels that are available under the MAC Switch driver.

It assures network availability even when problems occur with single network connections. MAC Switch driver design features include the following ones:

- Single virtual network view of multiple physical network connections.
- Alternate use of the physical network connections to optimize communications (allowing automatic load balancing between the physical networks).
- Monitoring of the status of the physical channels.

The MAC Switch driver sends and checks for a special packet of information, called the keepalive packet. If the keepalive packet is not received on a physical channel after some time, which can be configured with `mswconfig`, the MAC Switch driver marks that channel offline and stops sending packets via that channel. When the associated error condition is corrected, the MAC Switch driver automatically returns the channel to online state.

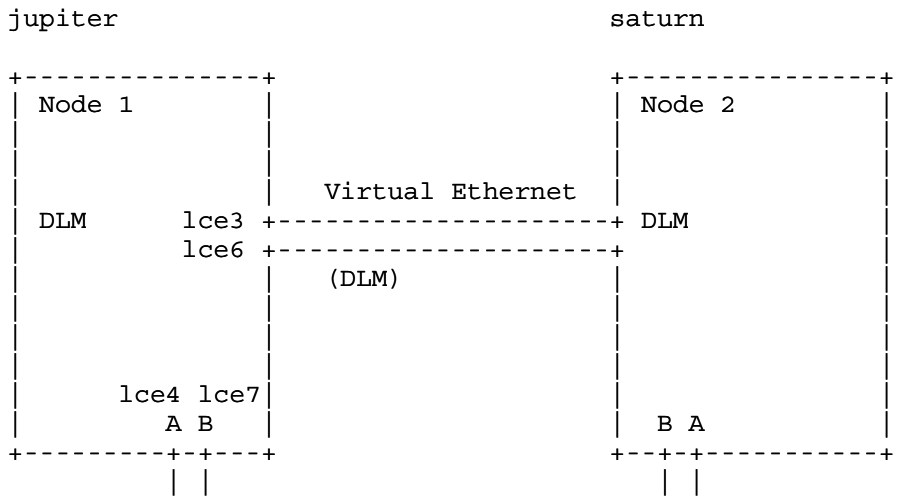
Before the MAC Switch driver can be used by a network subsystem, it must be configured. This consists of first creating the MAC Switch driver configuration file `/etc/mswtab`, and then using the `mswconfig` utility to configure the MAC Switch driver according to the `/etc/mswtab`. For details and complete syntax of `mswtab` and `mswconfig`, refer to the `mswconfig(8)` and `mswtab(4)` manual pages.

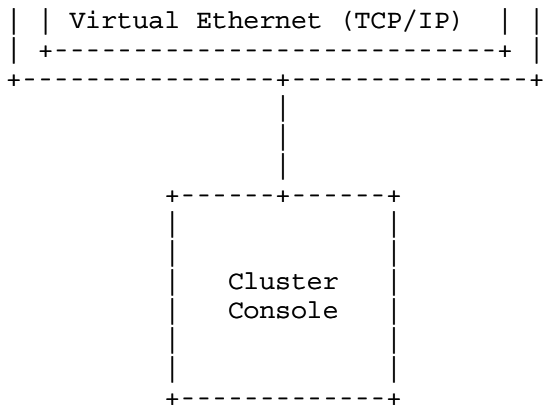
The MAC Switch driver will be automatically configured during system initialization by the startup script `/etc/init.d/SImsw`. In the output of `netstat -i` you will find "none" in the rows network and address for the physical pieces of a Virtual Ethernet. To determine which physical pieces belong to a Virtual Ethernet `sw<x>`, refer to the file `/etc/mswtab`.

The MAC Switch driver ist only released for Siemens Nixdorf layered products like the Distributed Lock Manager (DLM).

EXAMPLES

As an example we show how to administer the following configuration on Node 1 (`jupiter`):





I. Two (or more) physical LAN interfaces, e.g. `lce4` and `lce7`. For these two physical LAN interfaces, we want to administer a Virtual Ethernet interface with TCP/IP.

II. Two (or more) physical LAN interfaces, e.g. `lce3` and `lce6`. For these two physical lan interfaces, we want to administer a Virtual Ethernet interface for the DLM.

The administration of this typical configuration consists of creating the following configuration files:

```
/etc/mc2tab
/etc/mswtab
/etc/default/inet
/etc/clustertab
```

for Node 1 (and of course in the same manner for Node 2).

1. In our example, the format of the `/etc/mc2tab` file is as follows:

```
lce3
lce4
lce6
lce7
```

2. In our example, the format of the `/etc/mswtab` file is as follows:

```
sw0     2       8:0:6:5:52:a2
        lce     3
        lce     6
sw1     2       8:0:6:5:52:a3
        lce     4
        lce     7
```

3. In our example, we must add the following lines to the file `/etc/default/inet`:

```
EXTIF<X>=sw1:45.45.45.101:netmask:0xff000000:broadcast:45.255.255.255
INTERFACES="$STANDARDIF ..... $EXTIF<X>"
```

That's all to configure TCP(UDP)IP for `sw1`.

4. In our example, the format of the `/etc/clustertab` file is as follows:

```
                                            0 means DLM uses sw0
                                            |
                                           `.´
1       jupiter         /dev/msw/swether    0       8:0:6:5:52:a2
2       saturn          /dev/msw/swether    0       8:0:6:5:53:f4
```

NOTES

If a cluster console is used for the TCP/IP Virtual Ethernet, it must be connected to piece 0 of the Virtual Ethernet. To determine which `lce` board corresponds to Piece 0, use `mswconfig –l`.

```
jupiter.root:/> mswconfig -l sw1
```

```
Listing MAC Switch interface  sw1
sw0: Address:  8:0:6:5:52:a3 2 Piece(s) ONLINE
(Heartbeat ENABLED : Interval = 200 Secs : Misses = 6)
Piece 0: lce  4       Online (21)
Piece 1: lce  7       Online (21)
```

In our example, we must connect the cluster console to the physical LAN of `lce4`.

RESTRICTIONS

On the MAC interface, only the Ethernet protocol is supported (not LLC1 or SAP-SNAP).

The startup script `/etc/init.d/SImsw` resp. `mswconfig -c` does not configure the MAC Switch driver if there is an error on the piece 0 interface (e.g. board doesn't exist). In this case, you have two possibilities to start the Virtual Ethernet – either you repair this interface or you change the interface lines in `/etc/mswtab` to get a working interface as piece 0.

SEE ALSO

clustertab(4), mc2tab(4), mswtab(4), mswconfig(8).

Fujitsu Siemens Computers Add-on Reference Manual

sih(7)
RM400 only

NAME

`sih` - controller for HDLC devices

DESCRIPTION

HDLC devices are operated via the SIH controller (Serial Interface multiplexer inHouse). An SIH controller has six lines (HDLC strings) at its disposal, to which devices can be connected by means of both multipoint and point-to-point connections.

The connection is made via the interface converters SW8 and SW8R. Six inhouse lines are available for the SW8, while the SW8R has four V24 lines and two inhouse lines at its disposal.

Up to 32 devices with any link addresses (0-255) can be operated on each line. However, the total number of devices running simultaneously on all six lines must not exceed 96.

Two drivers (SVH and SIH) are required for the operation of the SIH controller. The SVH driver is used for loading the controller software and has no further significance for applications.

The SIH driver provides multi and single devices, as well as admin, load and debug devices for applications. The SIH driver provides a TTY interface as a system call interface for the individual devices.

The SIH driver is available in two incarnations, each with its own major device number.

The major device number is defined during installation of the driver software, and can be found in the sixth column of the `/etc/conf/cf.d/mdevice` file.

1) `sihm` driver □
The 16 DPTG2 channels are operated via the `sihm` driver.

The minor device numbers are used in accordance with the following algorithm:

`sihm` driver			
Bit	17□□□16□□□15□□□14 □□□13□□12	11□□□10□□□09□□□08□□□07□□ □06□□□05□□□04	03□□□02□□□0 1□□□00
	Line No.	Link-Adresse	Kanal-Nr.
	0...63	0...255	0...15

2) `sihs` driver □
The debug device, admin device, load device, and single device are operated via the `sihs` driver.

The minor device numbers are used in accordance with the following algorithm:

`sihs` driver			
Bit	17□□□16□□□15□□□14 □□□13□□12	11□□□10□□□09□□□08□□□07□□ □06□□□05□□□04	03□□□02□□□01□ □□00
	Line No.	Link-Adresse	Kanal-Nr.
	0...63	0...255	0...15
			0 = Single-Dev.□ 1 = Admin.-Dev.□ 2 = Load-Dev.□ 3 = Debug-Dev.□ 4 - 15 frei

The line numbers are allocated independently of the connections available from a controller.

SEE ALSO

hdlcio(7), hios(7), termio(7).

Fujitsu Siemens Computers Add-on Reference Manual

sraid(7)
RM400 only

NAME

`sraid -` SCSI RAID interface on an RM400

SYNOPSIS

`/dev/ios` *b*`/[ r ]sraid`*bcullsp*

DESCRIPTION

`sraid` provides SCSI magnetic RAID devices. IOS SCSI RAID devices are accessed through the special files in the directory `/dev/ios0`. The naming convention for all IOS RAID devices is as follows (all numbers are specified in decimal):

`/dev/ios`*b*`/[`r`]sraid`*bcullsp*

b Bus number: always 0.

`r` Designates the raw (character) device.

c The decimal controller number of the SCSI host adaptor. The controller number is either 0 or 1 on bus 0 or 0 to 3 on bus 1.

u The device's unit number (address). The address may range from 0 to 6.

l The logical unit number (LUN). This number may range from 0 to 6.

p Designates the LUN slice (partition) (0 to 15).

MAJOR-MINOR DEVICE NUMBER

The major-minor device number can be determined via `/sbin/autoconf –d` *device_name*. The meaning of the number returned is described below.

The major-minor number is a 32-bit number formatted as follows:

BBBMMMMM MMMMMMaa aaaccbdd duuupppp

B Bus number: 0 or 1.

M The major device number.

a The host adaptor number on the specified bus, always 0.

c The host adaptor's controller number.

b The SCSI bus the device is attached to on the specified host adaptor. Since each host adaptor has only one SCSI bus, *b* is always 0.

d The SCSI device address (0 to 6) on the specified SCSI bus.

u The SCSI Logical Unit Number of the specified SCSI device, if supported. Most devices do not use LUN's, therefore bits 4-6 will usually be 0.

p The partition number (0 to 15).

ERROR DESCRIPTION

The error messages displayed for the IOS RAID drives have the following formats:

```
ios0/sraid010l4s1: Write Extended 32 <seek> [3506:0:0] <dkblk> 721024 <pblk> 4824
256
     <status> 0b950200 (hard 1/1) Aborted Command - A drive channel was reset
     (RAID) Extended Sense Bytes 13-101 for lpb at 0xc04d5c00
     *  00020000 00300002 02450000 00000402 00040000 00031b11 00001202 2a800049
     *  9cc00000 20000000 0f315433 35303130 31323100 00000000 00303130 30010400
     *  00354434 3143525a 32382020 20202028 43292044 45430000 05000000
CMN: WARNING: cmn 38477  ios0/sraid010l4 Field Replaceable Unit (FRU) belongs to:
```

```
DRIVE GROUP
     Failure report for drive at controller SCSI channel 3, SCSI ID 5:
     List of failed drive components:
Time 14:59:16   up  1:55
CMN: WARNING: cmn 52643  ios0/sraid010l4 state changed to Degraded RAID5 Lun
ios0/sraid010l4s1: Write Extended 32 <seek> [3506:0:0] <dkblk> 721024 <pblk> 4824
256
     <status> 063f0200 (hard 2/2) Unit Attention - Caution: drive replace require
d!
     (RAID) Extended Sense Bytes 13-101 for lpb at 0xc04d5c00
     *  80350000 00000001 00000000 00000000 00000000 00000000 00000400 2a800049
     *  9cc00000 20000000 0f315433 35303130 31323100 00000000 00303130 30020401
     *  35124434 3143525a 32382020 20202028 43292044 45430000 05000000
```

The second error message will be described below:

`ios0/sraid010l4s1:`
: The IOS RAID drive in error. In this case, the RAID LUN located logical controller 1, drive 0, lun 4, in partition 1.

`<cmd>`
: The SCSI command in error. In this case, a write of 32 physical sectors was attempted.

`<seek>`
: The seek address for the write described by [*cylinder*:*head*:*sector*]. Note that the seek address is determined in one of two ways. If the SCSI request sense command extended sense information bytes are valid, then this value is decoded into cylinder, head, and sector values. Otherwise, the start address specified in (this case by) the SCSI write command is decoded. In either case, the seek address does not take into account any remapping that might have been done previously by `format`. So remapping is not based upon the seek address.

`<dkblk>`
: The 512-byte LUN block start number (offset from the current partition). In this case the eight sector write started at block 16 in partition 8.

`<pblk>`
: The physical LUN block number. Note that the physical LUN block number is determined in one of two ways. If the SCSI request sense command extended sense information bytes are valid, then this value is used. Otherwise, the start address specified in (this case by) the SCSI write command is used. Use this value for specifying a start address in `format`.

`<status>` *KKAASSCC*
: The IOS composite status code. The composite status code has the following format:

  KK The SCSI Sense Key value.

  AA The SCSI additional Sense Key value.

  SS The SCSI status returned to the SCSI host adaptor.

  CC The IOS host adaptor error code.

`(hard 1/1)`
: The error type, followed by the retry count. There are two error types: hard and soft. Soft errors are retried. Unrecoverable soft errors and errors that should not be retried are classified as hard. The retry count is expressed as the number of retries divided by the maximum retries allowed. In the case of hard errors, the value is always 1/1.

`Unit Not Ready`
: The decoded SCSI Sense Key value message.

`LUN not ready, START UNIT required`
: The decoded SCSI Additional Sense Key value message.

`(RAID)`
: The device type (RAID).

`Extended Sense Bytes`
: The byte address of the saved extended sense bytes displayed in the next lines after this message. In this case, bytes 12 to 135 of the MS12 LUN's extended sense data were saved after sending the SCSI request sense command after the error on the write SCSI command.

`lpb at`
: The kernel address of the local parameter block for the SCSI write command issued to the RAID LUN.

`* 80350000 ....`
: The extended sense data. The amount of data displayed depends upon the number of saved extended sense bytes specified for the device type's profile in `/etc/conf/pack.d/sri/space.c`.

SCSI RAID CONFIGURATION

SCSI devices and RAID LUNs are identified and initialized during boot time in the SCSI drive probe. A RAID LUN is not actually configured, however, until it has been opened. During system boot, a RAID device is identified solely on the basis of its SCSI inquiry name. Once a LUN has been opened, and SCSI geometry information has been retrieved, the device is configured. RAID LUN configuration may be checked at any time via the `autoconf` command. Below is sample `autoconf -l` output:

```
ios0/sraid070l0 MS12(4087MB) -- Optimal RAID5 LUN
ios0/sraid070l1 MS12: -- not configured
ios0/sraid070l2 MS12(3067MB) -- partition not set
```

Following the LUN device name are two fields: the type and the current LUN state. So, `ios0/sraid070l0` has a type name of `MS12`, and a LUN state of optimal RAID5. LUNs which haven't been configured yet display a shortened form of the SCSI device type name followed by an unconfigured message. RAID LUNs which are partially configured (i.e. `dkindex` hasn't been run yet), will show `partition not set`. See `dkindex(8)` and `dkpsdfinit(8)` for more information on configuring LUNs.

After drive spin up, the LUN format, geometry and logical unit parameters are retrieved through the mode sense SCSI command. This step occurs when the drive is first opened. The LUN format and geometry parameters are used to determine the physical characteristics of the LUN (for example, cylinders, heads, sectors, skew, and so forth). The logical unit parameters are used to determine the RAID level, the current state of the LUN and the number of physical LUN which are displayed by `autoconf` or used internally in the kernel. The MS12 supports the RAID Levels: 0,1,3 and 5. This information is used for error handling in the driver and displayed by `autoconf`. The LUN state can have one of the following: `Optimal`, `Degraded`, `Reconstructing` or `Dead`. A degraded or dead LUN requires manually intervention, which can be done with `apr(1M)` and replacing a failed physical drive.

The reserved cylinder count is further adjusted by one cylinder for a reserved cylinder in the data area. This area contains the disk index structure [see `psdf(4)`], and is read at configuration time. The command `dksetup` needs to be run to initialize a RAID LUN for use.

Finally, one more reserved cylinder for a CE cylinder in the data area is added. Since the native CE cylinder requires many gyrations to access, or is inaccessible to the mortal user, a CE cylinder in the data area is reserved for the Format Information Table produced by `format`. This contains information as to when the RAID LUN was last formatted, drive serial number, and basic geometry information. The defect list is maintained by the SCSI RAID controller, and is not stored physically on the RAID LUN.

SCSI RAID NOTES

The physical sector size is not necessarily 512 bytes and is dependent on the RAID level and the number of physical disks. The SCSI RAID driver converts the block count into the number of sectors required for a 512 byte transfer. When errors occur both the physical block and the system block size are printed. The formatter expects that the physical block will be provided, should a user wish to map out a bad block.

The LUNs of the RAID system can be configured and examined by `acf(1M)` . Use the `autoconf` command to verify any configuration changes.

SEE ALSO

`acf(1M)` , `apr(1M)` , `psdf(4)` , `autoconf(8)` , `dkindex(8)` , `dkpart(8)` , `dkpsdfinit(8)` ,

dksetup(8), format(8), setinfo(8).

Fujitsu Siemens Computers Add-on Reference Manual

sraid(7)
RM600 only

NAME

`sraid` - SCSI RAID interface on an RM600

SYNOPSIS

`/dev/ios` *b*`/[ r ]sraid`*ccu*`l`*x*`s`*p*

`/dev/iof0/ [ r ]sraid`*cc*`.`*nnnnn*`.`*x*`s`*p*

DESCRIPTION

`sraid` provides SCSI magnetic RAID devices. IOS SCSI RAID devices are accessed through the special files in the directory `/dev/ios`*b*. The naming convention for all IOS RAID devices is as follows (all numbers are specified in decimal):

`/dev/ios`*b*`/[r]sraid`*ccu*`l`*x*`s`*p*

b Bus number: always 0.

`r` Designates the raw (character) device.

cc The logical controller number. The controller number is any 2-character decimal number between 00 and 99. The number indicates which SCSI bus the device is connected to on which host adaptor. Physical controllers are assigned to the controller numbers in `/etc/controller`.

u The device's unit number (address). The address may range from 0 to 7. It is important that the device address does not conflict with the SCSI ID of the Host Adaptor port is it connected to.

x The number of the logical unit (LUN). This is a number in the range 0 to 7 for SCSI devices or 0 to 4095 for fiber channel devices.

p Designates the LUN slice (partition) (0 to 15).

If a Raid system is connected via fiber channel (FC), access is possible using the following names:

`/dev/iof0/[r]sraid`*cc*`.`*nnnnn*`.`*x*`s`*p*

The addressing is the same as in the familiar SCSI world. The *cc*, *p* and *x* parameters have the same meaning as in the case of SCSI Raid systems.

nnnnn is a value which is generated from the universally unique name for FC devices.

This name will continue to be used in error messages and statistics.

In addition, it is possible to address a disk by means of the serial number:

`/devices/[r]`*serial_number*

`/devices/[r]`*serial_number* `.priv`

Disks can be addressed by serial numbers in this way independently of their position; this simplifies administration, for example in the case of observe or multihost configurations. This addressing is used by Vdisk Lite and applies for SCSI and FC disks.

In contrast to earlier addressing, there are no longer16 partitions here: `/devices/[r]`*serial_number* refers to the usable data area on a disk and `/devices/[r]`*serial_number* `.priv` refers to the area reserved for `statesave` disks.

MAJOR-MINOR DEVICE NUMBER

The major-minor device number can be determined via `/sbin/autoconf -d` *device_name*. The meaning of the number returned is described below.

The major-minor number is a 32-bit number formatted as follows:

BBBMMMMM MMMMMMcc ccccc`0`*dd duuupppp*

B Bus number: always 0.

M The major device number.

c The logical controller number of the host adaptor. This can be any number between 00 and 99.

`0` Not used, 0.

d The SCSI device address (0 to 7) on the specified SCSI bus.

u The SCSI Logical Unit Number of the specified SCSI device, which is supported currently only by this driver.

p The partition number (0 to 15).

DEVICE(CLASS)NUMBER FOR FIBER CHANNEL DISKS

In contrast to SCSI disks, a different device number is used here. In addition, these disks do not have a fixed assignment of bits for the controller numbers and the SCSI ID; the device number is assigned dynamically and is only known to the device file system (`devfs`).

ERROR DESCRIPTION

The error messages displayed for the IOS RAID drives have the following formats:

```
ios0/sraid010l4s1: Write Extended 32 <seek> [3506:0:0] <dkblk> 721024 <pblk> 4824
256
     <status> 0b950200 (hard 1/1) Aborted Command - A drive channel was reset
     (RAID) Extended Sense Bytes 13-101 for lpb at 0xc04d5c00
     *  00020000 00300002 02450000 00000402 00040000 00031b11 00001202 2a800049
CMN: WARNING: cmn 38477  ios0/sraid010l4 Field Replaceable Unit (FRU) belongs to:
DRIVE GROUP
     Failure report for drive at controller SCSI channel 3, SCSI ID 5:
     List of failed drive components:
Time 14:59:16   up  1:55
CMN: WARNING: cmn 52643  ios0/sraid010l4 state changed to Degraded RAID5 Lun
ios0/sraid010l4s1: Write Extended 32 <seek> [3506:0:0] <dkblk> 721024 <pblk> 4824
256
     <status> 063f0200 (hard 2/2) Unit Attention - Caution: drive replace require
d!
     (RAID) Extended Sense Bytes 13-101 for lpb at 0xc04d5c00
     *  80350000 00000001 00000000 00000000 00000000 00000000 00000400 2a800049
```

The second error message will be described below:

`ios0/sraid010l4s1:`
: The IOS RAID drive in error. In this case, the RAID LUN located logical controller 1, drive 0, lun 4, in partition 1.

`<cmd>`
: The SCSI command in error. In this case, a write of 32 physical sectors was attempted.

`<seek>`
: The seek address for the write described by [*cylinder*:*head*:*sector*]. Note that the seek address is determined in one of two ways. If the SCSI request sense command extended sense information bytes are valid, then this value is decoded into cylinder, head, and sector values. Otherwise, the start address specified in (this case by) the SCSI write command is decoded. In either case, the seek address does not take into account any remapping that might have been done previously by `format`. So remapping is not based upon the seek address.

`<dkblk>`
: The 512-byte LUN block start number (offset from the current partition). In this case the eight sector write started at block 16 in partition 8.

`<pblk>`
: The physical LUN block number. Note that the physical LUN block number is determined in one of two ways. If the SCSI request sense command extended sense information bytes are valid, then this value is used. Otherwise, the start address specified in (this case by) the SCSI write command is used. Use

this value for specifying a start address in `format`.

`<status>` *KKAASSCC*
: The IOS composite status code. The composite status code has the following format:

KK The SCSI Sense Key value.

AA The SCSI additional Sense Key value.

SS The SCSI status returned to the SCSI host adaptor.

CC The IOS host adaptor error code.

`(hard 1/1)`
: The error type, followed by the retry count. There are two error types: hard and soft. Soft errors are retried. Unrecoverable soft errors and errors that should not be retried are classified as hard. The retry count is expressed as the number of retries divided by the maximum retries allowed. In the case of hard errors, the value is always 1/1.

`Unit Not Ready`
: The decoded SCSI Sense Key value message.

`LUN not ready, START UNIT required`
: The decoded SCSI Additional Sense Key value message.

`(RAID)`
: The device type (RAID).

`Extended Sense Bytes`
: The byte address of the saved extended sense bytes displayed in the next lines after this message. In this case, bytes 12 to 135 of the MS12 LUN's extended sense data were saved after sending the SCSI request sense command after the error on the write SCSI command.

`lpb at`
: The kernel address of the local parameter block for the SCSI write command issued to the RAID LUN.

`* 80350000 ....`
: The extended sense data. The amount of data displayed depends upon the number of saved extended sense bytes specified for the device type's profile in `/etc/conf/pack.d/sri/space.c`.

SCSI RAID CONFIGURATION

SCSI devices and RAID LUNs are identified and initialized during boot time in the SCSI drive probe. A RAID LUN is not actually configured, however, until it has been opened. During system boot, a RAID device is identified solely on the basis of its SCSI inquiry name. Once a LUN has been opened, and SCSI geometry information has been retrieved, the device is configured. RAID LUN configuration may be checked at any time via the `autoconf` command. Below is sample `autoconf -l` output:

```
ios0/sraid070l0 MS12(4087MB) -- Optimal RAID5 LUN
ios0/sraid070l1 MS12: -- not configured
ios0/sraid070l2 MS12(3067MB) -- partition not set
```

Following the LUN device name are two fields: the type and the current LUN state. So, `ios0/sraid070l0` has a type name of `MS12`, and a LUN state of optimal RAID5. LUNs which haven't been configured yet display a shortened form of the SCSI device type name followed by an unconfigured message. RAID LUNs which are partially configured (i.e. `dkindex` hasn't been run yet), will show `partition not set`. See `dkindex(8)` and `dkpsdfinit(8)` for more information on configuring LUNs.

After drive spin up, the LUN format, geometry and logical unit parameters are retrieved through the mode sense SCSI command. This step occurs when the drive is first opened. The LUN format and geometry parameters are used to determine the physical characteristics of the LUN (for example, cylinders, heads, sectors, skew, and so forth). The logical unit parameters are used to determine the RAID level, the current state of the LUN and the number of physical LUN which are displayed by `autoconf` or used internally in the kernel. The MS12 supports the RAID Levels: 0,1,3 and 5. This information is used for error handling in the driver and displayed by `autoconf`. The LUN state can have one of the following: `Optimal`, `Degraded`, `Reconstructing` or `Dead`. A degraded or dead LUN requires manually intervention, which can be done with

apr(1M) and replacing a failed physical drive.

The reserved cylinder count is further adjusted by one cylinder for a reserved cylinder in the data area. This area contains the disk index structure [see psdf(4)], and is read at configuration time. The command dksetup needs to be run to initialize a RAID LUN for use.

Finally, one more reserved cylinder for a CE cylinder in the data area is added. Since the native CE cylinder requires many gyrations to access, or is inaccessible to the mortal user, a CE cylinder in the data area is reserved for the Format Information Table produced by format. This contains information as to when the RAID LUN was last formatted, drive serial number, and basic geometry information. The defect list is maintained by the SCSI RAID controller, and is not stored physically on the RAID LUN.

SCSI RAID NOTES

The physical sector size is not necessarily 512 bytes and is dependent on the RAID level and the number of physical disks. The SCSI RAID driver converts the block count into the number of sectors required for a 512 byte transfer. When errors occur both the physical block and the system block size are printed. The formatter expects that the physical block will be provided, should a user wish to map out a bad block.

The LUNs of the RAID system can be configured and examined by acf(1M) . Use the autoconf command to verify any configuration changes.

SEE ALSO

acf(1M) , apr(1M) , psdf(4) , devfs(7) , vdisklite(7) , autoconf(8) , dkindex(8) , dkpart(8) , dkpsdfinit(8) , dksetup(8) , format(8) , setinfo(8) .

System Maintenance Procedures(8)

cdrom(8)

NAME

cdrom - administration program for CD-ROM drive

SYNOPSIS

cdrom [–f *device*] {lock | unlock | eject | slow | fast}

DESCRIPTION

The cdrom command can be used for administering the CD-ROM drive.

OPTIONS

–f *device*
: This option specifies the character special device of the CD-ROM drive. The default is /dev/ios0/rsdisk006s0.

lock
: This option prevents the CD-ROM being removed from the CD-ROM drive.

unlock
: This option releases the CD-ROM.

eject
: If unmount is not functioning correctly and the CD-ROM cannot be removed, this option ejects the CD-ROM from the drive.

slow
: You use this option to set the drive to operate at minimum speed. In rare cases, it is not possible to read self-written CD-ROM media in high-speed drives. You can use this option in this instance to reduce the read speed of the drive.

fast
: You use this option to set the drive to operate at maximum speed. The drive is generally operated at maximum speed. If you reduced the speed of the drive with the slow option, you have to reset the

drive to normal speed with the `fast` option as soon as the need for a reduced speed is eliminated.

FILES

`/dev/ios0/rsdisk[00-31][0-7]s0`

SEE ALSO

`mount`(1M) , `umount`(1M), `mount`(1M-hs) , `hs`(7) .

Fujitsu Siemens Computers Add-on Reference Manual

dlmconfig(8)
SIDLM

NAME

`dlmconfig` - Distributed Lock Manager configuration andadministrative utility

SYNOPSIS

`/sbin/dlmconfig` [`–bdgilprxmTtWw` *value*] [`–CcMNhKBQSsuv`] [*config_file*]

DESCRIPTION

The `dlmconfig` utility is an administrative utility to configure and control the Distributed Lock Manager (DLM). It allows both domain-specific structure allocation and global configuration for DLM parameters, such as deadlock detection, rebuild process, and remaster activity. In addition, it provides a global control mechanism for cluster configuration of the DLM.

OPTIONS

Domain-specific structure allocation options to `dlmconfig` are described below:

`–b`*max_buffer*
: Specify the maximum number of buffer blocks to be allocated.

`–d`*domain_num*
: Specify the domain number for the configuration changes. This number is substituted for the `X` in the string `/dev/DBlckXctl` to arrive at the device name. The default number is 0, or the whole device name is defined by the `LKMGRCTL` environment variable if that environment variable is set.

`–g`*max_group*
: Specify the maximum number of group blocks to be allocated.

`–i`*max_info*
: Specify the maximum number of info blocks to be allocated.

`–l`*max_lock*
: Specify the maximum number of lock blocks to be allocated.

`–p`*max_proc*
: Specify the maximum number of process blocks to be allocated.

`–r`*max_res*
: Specify the maximum number of resource blocks to be allocated.

`–x`*max_xact*
: Specify the maximum number of transaction blocks to be allocated.

Global configuration options to `dlmconfig` are described below:

`–C` Display current configuration information, current DLM state, and node connection status per domain.

`–c` Display current configuration information in a short format.

`–M` Enable remastering of all resources, regardless of activity.

`–m`*threshold*
: Specify the remaster activity threshold level.

`–N` Direct `dlmconfig` to configure the network and the cluster. The DLM adds 180 internal buffer blocks for each other node in the cluster every time the network is configured.

`–T`*deadtime*
: Specify the interval in seconds between successive deadlock searches. The default is 1 second.

`–t`*timeout*
: Specify the network timeout interval in seconds. The default is 28 seconds.

–W*deadwait*
: Specify the time in seconds DLM waits before initiating a search to resolve possible deadlock. The default is 10 seconds.

–w*waittime*
: Specify the wait time in seconds for the other nodes to configure into the network. The default is 5 seconds. The system can implicitly add an additional amount of time to the specified value depending on the number of nodes in the cluster.

Global control options are described below:

–h Generate a usage message.

–K Sends SIGKILL to all locally connected clients.

–B Distributed Lock Manager configuration manually.

–Q Put the node in a quiescent state; that is, initiate remastering of all master resources held on this node.

–S Stop this node from continuing participation in the DLM operations.

–s Start or restart the cluster configuration process.

–u Display statistical information. Among other statistics you will see "Allocation statistics". These show for each of the resource, lock, buffer, process, group, transaction, and info blocks the maximum number of allocatable blocks (MAX) as configured by dlmconfig, the current number of allocated blocks (NUM), the maximum number of blocks that were ever allocated at one point in time (AMAX), and the number of allocation failures (AFAIL). If AFAIL is nonzero, or if AMAX comes close to MAX, you should increase the number of blocks. On the other hand, if AMAX is much smaller than MAX, you may consider to choose smaller values next time you reboot the kernel and configure DLM, in order to save kernel memory space.

–v Specify verbose mode.

The –b, –g, –i, –l, –p, –r, and –x options define additional structure allocations internal to the DLM. They specify the different limits that the DLM can handle. For example, the maximum number of local clients is controlled by the –p option. However, any request to lower the limits is ignored.

The –N option is used when the cluster network is to be configured or reconfigured. Additionally the local node is started. Once the DLM is in the running state, the node IDs cannot be changed. However, new nodes can be added into the cluster, or an existing node can be deleted.

The –M, –m, –T, –t, –W, and –w options control the functional attributes of the DLM. The default values of these attributes are the recommended values. However, under some special circumstances, they can be altered through the use of these options.

The –K, –Q, –S, and –s options provide global control to the operations of the DLM. The –s option is used to start the local node, which causes a cluster reconfiguration as a result of the addition of the new node. The –S option is for shutting down the local node. The shutdown process is not successful if there are still local clients attached. If local clients do not detach on their own, the –K option can be used to kill off all locally attached clients. However, state information of locally opened resources cannot be guaranteed. To further preserve state information of resources, the –Q option is provided to initiate remastering of all local master resources. If the –K and –Q options are executed, then the node can be detached from the cluster through the –S option with the maximum amount of state information preserved.

ARGUMENTS

The dlmconfig utility configures the node's view of the cluster network using the *config_file*. If *config_file* is not specified, the default configuration file used is /etc/clustertab. The nodes to be configured into the cluster must all have the same *config_file*.

FILES

/etc/clustertab

DLM cluster configuration file

SEE ALSO

clustertab(4), dlm(7), autoconf(8).

Fujitsu Siemens Computers Add-on Reference Manual

dlmdump(8)

NAME

dlmdump - report status of DLM resources, locks and queues

SYNOPSIS

dlmdump {–r|–l} [–d *domain_num*] [–t {i|d|m|p}] [–i {test|ops}] [–V *sleeptime*]

DESCRIPTION

dlmdump traverses the DLM resource table and lock queues and prints the status of all resources or lock instances.

OPTIONS

–r Report the status of all resources.

–l Report the status of all lock instances.

–d *domain_num*
: Domain number for identifying a domain. Restrict report to a domain for resources or lock instances. Resources are configured for each domain, where the domain is identified by the number of the device used to open DLM. The report is issued for all domains by default.

–t Resource types. Restrict the report to one or more of the following resource types: i (inactive), d (directory), p (process), and m (master). The default is dmp (all except inactive resources).

–i Interpret resource names in the report as either ops or test. An ops resource name is reported as two letters for the lock type, followed by up to nine letters for the database name, then two integers for the lock number and lock class respectfully. Test resource names are reported as an integer. The default is ops.

–V *sleeptime*
: Visual mode. Sleep for *sleeptime* seconds and then echo a formfeed (CTRL-L) and repeat the report. The stdout stream is unbuffered. This allows dlmdump to pipe its output into display programs.

RESOURCE REPORT

The columns in the –r report have the following meanings:

Dom Number of the domain to which the resource belongs.

Typ The type of the resource; same as the –t option.

Cnt Number of locks open on the resource.

Ocnt For master resources, this is the number of pending requests to open a new lock instance. For process and directory resources, the current number of open locks.

GQ Number of locks in the "granted" queue for the resource.

CQ Number of locks in the "converting" queue for the resource.

Mas Node identification (as in /etc/clustertab) of the current "master" node for the resource.

Dir Node identification of the current "directory" node for the resource.

V Status of the "value block"; V means valid, N means there is no value block, and D means dubious.

F If this is Y, then the resource is frozen due to rebuild, deadlock, or remastering in progress.

NL CR CW PR PW EX
: These indicate the number of locks in the grant queue at each level.

Resource
: The name of the resource, interpreted according to the –i option. For the "ops" interpretation, the format is locktype.dbname.locknumber.lockclass.

LOCK REPORT

The columns in the `-l` report have the following meanings:

`Dom` Number of the domain to which the resource belongs.

`Pid` The process ID of the attached client. If the lock is a master lock (`Typ` is "M"), then this field is reported as "mast". In these cases, there will be additional lock instances reported for the associated local or process locks.

`Own` The node identification of the owner of the lock.

`Typ` `Typ` The type of the lock. Either inactive (`I`), process (`P`), master (`M`) or local (`L`).

`State`
Current state of the lock. Either opening (`OP`), granted (`GR`), converting (`CV`), openconverting (`OC`), canceling (`CA`), deleting (`DE`) or closing (`CL`).

`Lev` The current granted level of the lock.

`Opt` The lock options flags.

`Resource`
The name of the resource, interpreted according to the `-i` option.

CAVEATS

The resource queue traversal (and nested lock queue traversals) may fail on extremely active systems. Also, the lock report should print details of the associated transactions.

FILES

`/dev/kmem`
to read the kernel memory

`/stand/unix`
for the name list

`/etc/clustertab`
node id numbers

SEE ALSO

`dlm`(7), `dlmconfig`(8), `dlmqa`(8), `dlmstats`(8).

Fujitsu Siemens Computers Add-on Reference Manual

dlmconfig(8)
SPS

NAME

`dlmconfig -` Distributed Lock Manager configuration andadministrative utility

SYNOPSIS

`/sbin/dlmconfig [ –bdgilprxmTWw` *value* `] [ –CcMhKuv ]`

DESCRIPTION

The `dlmconfig` utility is an administrative utility to configure and control the Distributed Lock Manager (DLM). It allows both domain-specific structure allocation and global configuration for DLM parameters, such as deadlock detection, rebuild process, and remaster activity. In addition, it provides a global control mechanism for cluster configuration of the DLM.

OPTIONS

Domain-specific structure allocation options to `dlmconfig` are described below:

`–b` *max_buffer*
: Specify the maximum number of buffer blocks to be allocated.

`–d` *domain_num*
: Specify the domain number for the configuration changes. This number is substituted for the `X` in the string `/dev/DBlckXctl` to arrive at the device name. The default number is 0, or the whole device name is defined by the `LKMGRCTL` environment variable if that environment variable is set.

`–g` *max_group*
: Specify the maximum number of group blocks to be allocated.

`–i` *max_info*
: Specify the maximum number of info blocks to be allocated.

`–l` *max_lock*
: Specify the maximum number of lock blocks to be allocated.

`–p` *max_proc*
: Specify the maximum number of process blocks to be allocated.

`–r` *max_res*
: Specify the maximum number of resource blocks to be allocated.

`–x` *max_xact*
: Specify the maximum number of transaction blocks to be allocated.

Global configuration options to `dlmconfig` are described below:

`–C`
: Display current configuration information, current DLM state, and node connection status per domain.

`–c`
: Display current configuration information in a short format.

`–M`
: Enable remastering of all resources, regardless of activity. Per default only in some cases of activity on resources a remastering will be done for these resources.

`–m` *threshold*
: Specify the remaster activity threshold level. The default is 255 seconds.

`–T` *deadtime*
: Specify the interval in seconds between successive deadlock searches. The default is 1 second.

`–W` *deadwait*
: Specify the time in seconds DLM waits before initiating a search to resolve possible deadlock. The default is 10 seconds.

`–w` *waittime*

Specify the wait time in seconds for the other nodes to configure into the network. The default is 5 seconds. The system can implicitly add an additional amount of time to the specified value depending on the number of nodes in the cluster.

Global control options are described below:

`-h` Generate a usage message.

`-K` Sends `SIGKILL` to all locally connected clients.

`-u` Display statistical information. Among other statistics you will see "Allocation statistics". These show for each of the resource, lock, buffer, process, group, transaction, and info blocks the maximum number of allocatable blocks (`MAX`) as configured by `dlmconfig`, the current number of allocated blocks (`NUM`), the maximum number of blocks that were ever allocated at one point in time (`AMAX`), and the number of allocation failures (`AFAIL`). If `AFAIL` is nonzero, or if `AMAX` comes close to `MAX`, you should increase the number of blocks. On the other hand, if `AMAX` is much smaller than `MAX`, you may consider to choose smaller values next time you reboot the kernel and configure DLM, in order to save kernel memory space.

`-v` Specify verbose mode, used in conjunction with the `-b`, `-g`,... options to see, how much memory would be allocated. In verbose mode there is no real memory allocated.

The `-b`, `-g`, `-i`, `-l`, `-p`, `-r`, and `-x` options define additional structure allocations internal to the DLM. They specify the different limits that the DLM can handle. For example, the maximum number of local clients is controlled by the `-p` option. However, any request to lower the limits is ignored.

The `-M`, `-m`, `-T`, `-W`, and `-w` options control the functional attributes of the DLM. The default values of these attributes are the recommended values. However, under some special circumstances, they can be altered through the use of these options.

The `-K` option provides global control to the operations of the DLM. If local clients do not detach (during shutdown) on their own, the `-K` option can be used to kill off all locally attached clients.

SEE ALSO

`dlm(7)`, `autoconf(8)`.

Fujitsu Siemens Computers Add-on Reference Manual

dlmdump(8)

NAME

dlmdump - report status of DLM resources, locks and queues

SYNOPSIS

dlmdump {–r|–l} [–d *domain_num*] [–t {i|d|m|p}] [–i {test|ops}] [–V *sleeptime*]

DESCRIPTION

dlmdump traverses the DLM resource table and lock queues and prints the status of all resources or lock instances.

OPTIONS

–r Report the status of all resources.

–l Report the status of all lock instances.

–d *domain_num*
Domain number for identifying a domain. Restrict report to a domain for resources or lock instances. Resources are configured for each domain, where the domain is identified by the number of the device used to open DLM. The report is issued for all domains by default.

–t Resource types. Restrict the report to one or more of the following resource types: i (inactive), d (directory), p (process), and m (master). The default is dmp (all except inactive resources).

–i Interpret resource names in the report as either "ops" or "test". An "ops" resource name is reported as two letters for the lock type, followed by up to nine letters for the database name, then two integers for the lock number and lock class respectfully. Test resource names are reported as an integer. The default is "ops".

–V *sleeptime*
Visual mode. Sleep for *sleeptime* seconds and then echo a formfeed (CTRL-L) and repeat the report. The stdout stream is unbuffered. This allows dlmdump to pipe its output into display programs.

RESOURCE REPORT

The columns in the –r report have the following meanings:

Dom Number of the domain to which the resource belongs.

Typ The type of the resource; same as the –t option.

Cnt Number of locks open on the resource.

Ocnt For master resources, this is the number of pending requests to open a new lock instance. For process and directory resources, the current number of open locks.

GQ Number of locks in the "granted" queue for the resource.

CQ Number of locks in the "converting" queue for the resource.

Mas Node identification of the current "master" node for the resource.

Dir Node identification of the current "directory" node for the resource.

V Status of the "value block"; V means valid, N means there is no value block, and D means dubious.

F If this is Y, then the resource is frozen due to rebuild, deadlock, or remastering in progress.

NL CR CW PR PW EX
These indicate the number of locks in the grant queue at each level.

Resource
The name of the resource, interpreted according to the –i option. For the "ops" interpretation, the format is locktype.dbname.locknumber.lockclass.

LOCK REPORT

The columns in the `-l` report have the following meanings:

`Dom` Number of the domain to which the resource belongs.

`Pid` The process ID of the attached client. If the lock is a master lock (`Typ` is "M"), then this field is reported as "mast". In these cases, there will be additional lock instances reported for the associated local or process locks.

`Own` The node identification of the owner of the lock.

`Typ` The type of the lock. Either inactive (`I`), process (`P`), master (`M`) or local (`L`).

`State`
Current state of the lock. Either opening (`OP`), granted (`GR`), converting (`CV`), openconverting (`OC`), canceling (`CA`), deleting (`DE`) or closing (`CL`).

`Lev` The current granted level of the lock.

`Opt` The lock options flags.

`Resource`
The name of the resource, interpreted according to the `-i` option.

CAVEATS

The resource queue traversal (and nested lock queue traversals) may fail on extremely active systems.

FILES

`/dev/kmem`
to read the kernel memory

`/stand/unix`
for the name list

SEE ALSO

`dlmstat`(1M), `dlm`(7), `dlmconfig`(8).

Fujitsu Siemens Computers Add-on Reference Manual

dlmqa(8)
SIDLM

NAME

`dlmqa` - DLM exerciser

SYNOPSIS

`dlmqa` `-v|-t` [`-S`] [`-o`] [`-s` *seconds*] [`-i` *interpretation*] [`-r` *resname*[, *resname* ...] [`-l` *locktype*[, *locktype* ...] [`-c` *cycles*] [`-n` *iterations*]

`dlmqa` `-v|-t` [`-C` *hostname*] [`-o`] [`-s` *seconds*] [`-i` *interpretation*] [`-r` *resname*[, *resname* ...] [`-l` *locktype*[,*locktype* ...] [`-c` *cycles*] [`-n` *iterations*]

DESCRIPTION

The `dlmqa` utility provides a controlled way to verify the functionality and performance of the DLM and can be used to induce resource contention in active SIDLM clusters.

OPTIONS

The `dlmqa` tool performs DLM operations in up to four nested loops according to the following options:

`-n` *iterations*
: The outer-most loop; repeat everything *iterations* times.

`-r` *resname*[, *resname* ...]
: The resource names to use, interpreted according to the `-i` option (see below). The convert cycle loop (see `-l` below) is iterated serially for each named resource and then repeated according to the `-n` option. If you need to interleave convert cycles for several resources, then run several instances of `dlmqa`. This can be performed in lock-step between two `dlmqa` instances (on the same or different cluster nodes) using the `-S` and `-C` options (see below).

`-i` *interpretation*
: Use DLM resource name structures according to *interpretation*, which is either "ops" or "test" (the default). An example of an "ops" resource name is `BL.dbname.1234.1`, meaning the `BL` lock type in the `dbname` database, lock number 1234, lock class 1. "Test" resource names are simply integers converted from the argument string into an unsigned long. Note that `dlmstats(8)` understands both interpretations.

`-l` *lock*[, *lock* ...]
: The lock cycle. This is a comma separated list of lock levels, either in numeric or symbolic form. For example, `NL,EX,NL` is equivalent to `0,5,0`. After each resource lock is opened, the lock will be converted according to this cycle. The whole cycle will be repeated according to the `-c` option without closing the lock between iterations. If `-l` is not given, then the conversion cycle is of zero length, and thus `open – close` (or `openconvert – close`) will be tested without any intervening converts. If `-l` is not given then `-c` is ignored.

`-c` *cycles*
: Repeat the lock conversion cycle *cycles* times before closing. If *cycles* is negative, then the lock cycle will be repeated forever, thus keeping the resource open and the lock continuously converting.

`-o`
: On opening the lock for each resource, use an `openconvert` to the level of the first lock in the cycle. If the lock cycle is of zero length, then `openconvert` to level `NL`.

`-s` *seconds*
: Sleep for *seconds* after each convert operation. The default is not to sleep. This option is not very compatible with `-t`.

`-v`
: Verbose mode. Print a message for everything performed. Mutually exclusive with `-t`.

`-t`
: Timed mode. At the completion of the run, report the user plus system time (= CPU time) and the

elapsed real time. Mutually exclusive with `-v`.

`-S` Server mode. Run `dlmqa` in lock-step with a client (`-C`) instance of `dlmqa`. The server `dlmqa` must be started before the client `dlmqa`, which can (but need not) be on the same host, or even the same SIDLM cluster (TCP streams sockets are used on port 8001). If the client and server `dlmqa` instances are not on hosts from the same SIDLM cluster, then obviously there will be no resource sharing or lock contention.

`-C` *hostname*
Client mode. Run `dlmqa` in lock-step with the server currently waiting for a TCP connection on *hostname*. Both the client and server should be given arguments which will result in the same total number of DLM operations, though the lock cycle and/or resource names may be different. Care is required to avoid deadlock. For example, if you run the following, it will never terminate:

```
dlmqa -i test -r 0 -n 5 -l NL,EX,NL -S &
dlmqa -i test -r 0 -n 5 -l NL,EX,NL -C localhost
```

In this case, if the resource locked by the client is changed from 0 to 1, then there is no conflict.

CAVEATS

All calls to `libdlm` functions use the synchronous functions; there is currently no way to use the asynchronous versions. NULL value blocks are used for all DLM operations.

The DLM balances its load by choosing a node to master new resources in a pseudo random fashion that is based on the index of the resource-name-hash-bucket `mod` the number of nodes in the cluster), and then may re-master a particular resource at any time (depending on a number of conditions). This occasionally results in unexpected remote operations.

SEE ALSO

`dlm(7)`, `dlmconfig(8)`, `dlmdump(8)`, `dlmstats(8)`.

Fujitsu Siemens Computers Add-on Reference Manual

dlmstats(8)
SIDLM

NAME

`dlmstats -` report statistics on the distributed lock manager(DLM)

SYNOPSIS

`/sbin/dlmstats` [*global_options*] [*control_options*] [*report_option*] [*interval* [*count*]]

DESCRIPTION

The major use for `dlmstats` is to help identify and resolve contention for shared resources between clients of the DLM.

The options available for `dlmstats` fall into three groups; global options, control options, and report options. Global options are processed first and apply to all control and report options. Control operations are processed next. Any number of control options can be given and they are processed in the order specified. At most one report option is allowed and will be processed after all global and control options, if any.

If `dlmstats` is invoked without any options, a detailed usage report will be printed. All options are described in further detail below.

Before any report options can be used, statistics gathering must be enabled in the kernel. The `-L` control option is used for this; for example, `dlmstats -L1` will enable the coarse level of statistics on the local host. The command `dlmstats -c 4` will report coarse level statistics as a rate (DLM operations per second), sampled over four second intervals. These two commands can be combined; for example, `dlmstats -L1 -c 4`.

Global options are:

`-i ops|test`
: Specify the resource name interpretation. The default is `ops`, for Oracle Parallel Server.

`-n` *host*[**,** *host* ...]
: Apply all control and report options to all hosts in the comma-separated list. Note there is no space before or after each comma. If neither `-n` nor `-N` are given, the default is the local host. `dlmstats` is a client of `inetd`(1M) . When invoked, it opens a TCP/STREAMS socket to `in.dlmsd` (forked from `inetd`) on each host. To exchange the statistical information between the nodes, `in.dlmsd` should be configured as a child of `inetd` in the same way on all nodes of the cluster. We recommend the following entry in `/etc/inetd.conf`:

```
dlms stream tcp nowait root /usr/sbin/in.dlmsd in.dlmsd
```

and in `/etc/services`:

```
dlms   7998/tcp   # for dlmstats
```

`dlmstats` and `in.dlmsd` communicate using a simple protocol that effectively implements remote `ioctl`(2) system calls. The statistics that are returned by each host are aggregated over the intersection of resource name patterns. Resource patterns which are not in the resource table on *all* hosts are aggregated in the other (catchall) slot.

`-N`
: Apply all control and report options to all hosts present in `/etc/clustertab` (all nodes on the local cluster) on the local host.

Control options are:

`-L` *level*
: Change the level of statistics gathering in the kernel to level *level*. If *level* is `?`, then report the current level (but do not change it). Level 0 disables statistics collection. Level 1 collects coarse statistics (see `-c`). Level 2 collects medium statistics (see `-m`, `-r`, `-l` and `-u`). The performance overhead on the DLM associated with level 0 statistics collection is essentially zero. At level 1, it is approximately 2%, and at level 2, it is between 2% and 10% depending on the contents of the resource table and whether the

resource table miss cache is enabled.

−F *flagbits*
Set the low order bits on the statistics level. Presently, if *flagbits* is 1, then the resource cache will be enabled. If it is 0, the resource miss cache is disabled. See −M.

−Z Reset all statistics counts at the current level of collection to zero. This does not clear the resource table.

−R List the resource table contents.

−C Clear all entries in the resource table.

−M List the resource miss cache. Use this with −F to discover which resources are active and what the resource name patterns should be. See −A.

−A *resource_pattern*[, *resource_pattern* ...]
Add resource name patterns to the resource table. An OPS resource name pattern specification has the following syntax:

locktype[. *dbname*[. *locknumber*[. *lockclass*]]]

All fields are optional (defaulting to *), except *locktype*. Any field can be wild-carded using * (beware shell meta expansion). Any single character in *locktype* and *dbname* can be matched using ?. Note: *locknumber* and *lockclass* are represented internally as binary integers and hence cannot use ?. The "test" interpretation is simply an integer that is converted into a binary integer internally.

−D *slot*[, *slot* ...]
Delete one or more slot numbers from the resource table. Use −R to determine which resource name patterns occupy which slot numbers. Note: the last entry in the resource table can not be deleted. It is a catchall for matching resource names which do not match any other entry.

Report options are:

−c Report coarse level statistics. This report only contains statistics for DLM operation counts (or rates). There are no details for specific resources or lock types.

−m Report medium level statistics. This report tabulates resource table entries with DLM operation counts.

−u Report sorted statistics. This report contains a list of resource slots and lock conversions, sorted by highest operation count. The following command will report the most active DLM resources and lock conversions on all nodes in the cluster as a rate of DLM operations per second, sampled every eight seconds:

```
dlmstats -NL2 -u 8
```

The sort key considers delayed operations to be an order of magnitude more expensive than immediate operations.

−r *slot*
Report details of lock conversions (level from by level to) for the resource name pattern in slot *slot* of the resource table.

−l *locktype*
Report details of conversions to level *locktype* (from any level). *locktype* can be: NL, CR, CW, PR, PW, or EX.

−t Report Low Level Transport (LLT) statistics.

[*interval* [*count*]]
An optional interval and count causes the report to be repeated every *interval* seconds forever (or *count* times). Rather than report absolute counts, all statistics are reported as a rate per second over the interval.

BACKGROUND

Statistics about DLM operations are reported as either absolute counts, or if an *interval* is given, as rates (operations per elapsed second). All statistics are divided into two classes: immediate and delayed. The

precise semantics governing when an `immediate` or `delayed` operation is counted are specific to each type of operation and the current demand for the particular resource. Basically, an `immediate` operation is counted when the client can proceed without delay, for example when the DLM operation does not impede the progress of the client.

The kernel data structures for holding the statistics counters are of limited size and yet there may be hundreds of resources with active locks on any particular node. Consequently, a *resource pattern table* is maintained. This table contains resource name patterns (see `-A`). When a client such as OPS issues a DLM operation, the appropriate statistic is found using the DLM operation type, the existing and requested lock levels, and the resource table slot number. The last entry in the table is reserved as other for the cases where the resource involved in a DLM operation does not match any of the patterns in the table.

`dlmstats` is a distributed tool in that it can aggregate the statistics obtained from multiple hosts. This is an important feature because it allows an overview of the resource contention patterns on the whole cluster.

By default, `dlmstats` reports on resources in domain number 0. This can be changed by setting the `LKMGRCTL` environment variable [see `dlmconfig(8)`].

CAVEATS

The statistical aggregation relies on forming the intersection of all resource name patterns in the resource tables of all hosts. While these mechanisms cope with the same pattern occupying different slot numbers on different hosts, the process relies on binary string comparisons. For this reason, it is strongly suggested that all operations to manipulate resource name patterns be applied to all hosts at the same time. Always use `-N` and `-A` together.

FILES

`/etc/clustertab`
: DLM cluster configuration file

SEE ALSO

`inetd(1M)`, `clustertab(4)`, `dlm(7)`, `dlmconfig(8)`.

Fujitsu Siemens Computers Add-on Reference Manual

hszterm(8)

NAME

`hszterm -` control program for SY12/PXRC disk array controller

SYNOPSIS

`hszterm` [*options*] [*commands*]

DESCRIPTION

The `hszterm` command either starts a virtual terminal session with the SY12/PXRC or sends commands to the SY12/PXRC with the output of those commands going to stdout. All `hszterm` communication with the SY12/PXRC is done over the SCSI bus.

To run `hszterm` a device representing one of the units on the SY12/PXRC must be specified. This can be by using the `-f` option and providing the the path to the device.

If no additional arguments are provided, `hszterm` starts a virtual terminal session to the SY12/PXRC. The session and the program can be ended by entering the EOF character.

If additional arguments are provided, `hszterm` will uses the remaining arguments on the command line as commands to be passed to the HSZ. If the command to be passed consists of multiple words, it must be surrounded by quotes.

OPTIONS

`-f` *special*
: Character special device name of an SY12/PXRC logical units. If no logical units are defined, the bus and target number can be specified instead.

`-o` *logfile*
: Specifies the name of an optional log file. Messages sent to and received from the SY12/PXRC are appended to this file.

`-h`
: Print a brief help message.

`-v`
: Set level of verbosity. `hszterm` can be made to perform its actions with a great deal of, generally extraneous, output. This option allows setting the level of verbosity of this extra output.

`-l`
: Display all configuration-specific data relating to PXRC RAID controllers or magnetic disks. This information should be saved when the configuration is activated for the first time or changed. The data should not be stored on the RAID system, as it is needed to restore the current configuration, and thereby the data, if both controllers are lost.

RESTRICTIONS

The `hszterm` command restricts access to the SY12/PXRC programs that can be used with the `run` command. The set of allowed commands can be set using the environment variable `HSZTERM_RUN`. This is a comma separated list of the utilities the user would like to be able to run. If no list is specified the default list will consist of `CONFIG`, `FLS`, `FMU`, `CLCP` and `CLONE`. It will never allow running the utilities `CFMENU`, `VTDPY`, `DILX` or `TILX`, even when they are included in the list specified by `HSZTERM_RUN`.

DIAGNOSTICS

Most error messages are preceded by the string `HSZterm`. The errors message fall in three general categories; SCSI communication errors, virtual terminal errors and other errors.

`Getopt got confused`
: The library function `getopt(3C)` is used to parse the command line. This message is printed when an unrecognized option is used.

`Can't open %s: %s.`
: If `hszterm` is unable to open the `/dev/cam` or the special device file of the SY12/PXRC, this message

is printed. The system message corresponding to the error is provided.

`Can't open %s for logging: %s.`
If `hszterm` cannot open the log file, this message is printed.

`Can't close %s: %s.`
`hszterm` tries to close all the files it opens before it exits. If one of the files cannot be closed, this message is printed with the name of the file and system error message being given.

`No device selected. Select a device and try again.`
This message is printed when no device has been selected.

The following messages indicate errors in the virtual terminal protocol used to communicate with the SY12/PXRC.

`Device does not support Virtual Terminal protocol.`

`Failed to send response to remote program.`

`Failed to receive message.`

EXAMPLES

This example passes the command "show devices full" to the SY12/PXRC that is expected to correspond to the special device `/dev/ios0/rsraid040`.

```
# hszterm –f /dev/ios0/rsraid040 "show devices full"
```

This example limits the run list to `CONFIG`, `CLONE` and `FLS`.

```
# setenv HSZTERM_RUN CONFIG,CLONE,FLS
```

.fi

There can only be one `hszterm` session to an SY12/PXRC controller at a any time.

Fujitsu Siemens Computers Add-on Reference Manual

mswconfig(8)
SIDLM

NAME

`mswconfig -` configure Media Access Control (MAC) switch driver

SYNOPSIS

`mswconfig –h`

`mswconfig –l [ –i ] [ name ]`

`mswconfig {–c|–u} [ –p# ] [ –e [–t# ] [ –m# ]] [ –i ] [ name ]`

`mswconfig {–c|–u} [ –p# ] [ –d ] [ –i ] [ name ]`

`mswconfig –e [ –t# ] [ –m# ] [ {–c|–u} [–p# ]] [ –i ] [ name ]`

`mswconfig –d [ {–c|–u} [–p# ]] [ –i ] [ name ]`

DESCRIPTION

`mswconfig` is an administrative utility used to configure and manage the MAC switch driver configuration. The MAC switch driver combines redundant network connections (for example, one or more Ethernets) and provides a single virtual network view to the applications using the MAC switch driver. If one of the underlying network interfaces fails, that network interface is marked as `Offline` and the application packets are automatically switched to the other `Online` network interfaces under that MAC switch driver virtual interface.

Use the `mswconfig` command to perform the following actions on a MAC switch interface: configure, unconfigure, enable/disable heartbeat, change heartbeat interval and list the current configuration.

OPTIONS

`–h` Print the command usage.

`–l` List current MAC switch driver configuration.

`–c` Configure MAC switch driver virtual interface(s).

`–u` Unconfigure MAC switch driver virtual interface(s).

`–p#` Configure/unconfigure the given piece. Specify only with `–c` or `–u` options.

`–e` Enable heartbeat and set the time interval and heartbeat misses corresponding to the `–t` and `–m` options. If `–t` or `–m` are missing, the corresponding default values are used.

`–t` Heartbeat interval in seconds. Specify only with `–e` option. The default time interval is 4 seconds.

`–m` Heartbeat misses tolerated before changing the state of online piece to offline. Specify only with the `–e` option. The default value is 6 misses.

`–d` Disable heartbeat.

`–i` Use `stdin` for the `mswtab` file.

name Name of the MAC switch driver specified in the configuration file `/etc/mswtab`. If this parameter is not specified, the command applies to all MAC switch driver virtual interfaces defined in `/etc/mswtab`.

EXAMPLES

The following command configures the MAC switch driver virtual interface `sw0` using the configuration specified in `/etc/mswtab`:

```
mswconfig -c sw0
```

To configure just one piece (piece 0), enter the following command

```
mswconfig -c -p 0 sw0
```

To unconfigure just one piece (piece 1), enter the following command:

```
mswconfig -u -p 1
```

To enable the heartbeat (with a 3 second interval and 6 misses tolerated), use the following command:

```
mswconfig -e -t 3 -m 6
```

To list the current configuration of the MAC switch driver virtual interface `sw0`, enter the following command:

```
mswconfig -l sw0
```

The resulting display is similar to this:

```
listing MAC Switch interface
sw0: Address: 08:00:06:05:51:01 2 Piece(s) ONLINE
(Heartbeat ENABLED : Interval = 3 Secs : Misses = 6)
Piece 0: lce   1        Online (21)
Piece 1: lce   4        Online (21)
```

Using the previous configuration example, you would enter the following command to unconfigure piece 1:

```
mswconfig -u -p 1
```

Using the previous configuration example, you would enter the following command to list the current configuration:

```
mswconfig -l sw0
```

The resulting display is similar to this:

```
Listing MAC Switch interface
sw0: Address: 08:00:06:05:51:01 1 Piece(s) ONLINE
(Heartbeat ENABLED : Interval = 3 Secs : Misses = 6)
Piece 0: lce   1        Online (21)
Piece 1: lce   4        Not linked
```

RESTRICTIONS

`mswconfig -c` does not configure the MAC Switch driver, if there is an error on the piece 0 interface (e.g. board doesn't exist). In this case you have two possibilities to start the Virtual Ethernet – either you repair this interface or you change the interface lines in `/etc/mswtab` to get a working interface as piece 0.

FILES

`/etc/mswtab`

SEE ALSO

`ifconfig`(1M), `mc2tab`(4), `mswtab`(4), `strcf`(4).

Fujitsu Siemens Computers Add-on Reference Manual

rddmon(8)

NAME

rddmon - daemon for monitoring status changes in PXRC and PXRE RAIDsystems

SYNOPSIS

rddmon –s [–a *cycle_time*]

rddmon –H [–a *cycle_time*]

rddmon –t

DESCRIPTION

The rddmon RAID monitoring daemon is generally initiated at system startup with the S72rddmon start script. It cyclically calculates (default = 300 sec.) various statuses of the different PXRC and PXRE systems listed in /dev/autoconf.

Status changes are output as console messages and logged in a log file. Follow-up activities can then be initiated based on the log file entries using an action script. The overall objective is to provide information early on for the system support specialist to enable the problem to be rectified.

The rddmon daemon monitors the following statuses:

- LUN status (optimal/reconstructing/degraded/dead) of the PXRC and PXRE LUNs listed in /dev/autoconf.
- The components registered by the EMU (Environmental Monitoring Unit), i.e. power supplies, fans, temperature sensor and UPS (Uninterruptible Power Supply).
- The age of the cache battery is also calculated with PXRE systems; an appropriate message is output shortly before and after the expected life span has been reached.

OPTIONS

–s Starts the rddmon daemon.

–H Sends a SIGHUP signal to the active rddmon to initiate an immediate status check, for example following a configuration change.

If the rddmon daemon is not active, it is started with –H; in this case, the –a option is evaluated.

–a *cycle_time*

Polling cycle of the rddmon daemon; the default is 300 seconds. Only works with the –s and –H options, if a daemon is not yet active.

–t The active daemon is terminated; this has the same effect as the kill -9*rddmon_PID* command, but is more comfortable to use, for example if you want to vary the cycle time for the purpose of testing.

SIGNALS

The following signals can be used to influence the rddmon daemon during the life cycle:

SIGHUP

Initiates an immediate status check, see the –H option. If the rddmon daemon is not yet active, it is started; in this case, the –a option is also evaluated for setting the cycle time.

SIGTERM

Terminates the active daemon, see the –t option. This is only useful if you want to vary the cycle time for testing purposes.

OUTPUT

A complete and up-to-date list of log file entries generated by rddmon, including a description of errors and recommended actions, can be created with the

`getemm -c rdd`
command [see `getemm(1M)`].
Individual error descriptions can be queried with the
`getemm rdd:`*number*
command.
You should use SYSADM to view the log file.

EXIT STATUS

`0` Normal termination of parent process which starts the daemon and termination of the daemon following `rddmon -t`.

`1` Abnormal termination:

- The daemon could not be started
- Nothing to monitor in `/dev/autoconf`
- With `rddmon -t`: No daemon active

`2` Parameter error, user is not `root`.

FILES

This device node contains the RAID systems to be monitored:
`/dev/autoconf`
Action description file for LOG3:
`/opt/log3/filters/events.rdd`
Message catalogs for the `getemm` command:

```
/usr/lib/locale/EN_US.ASCII/LC_MESSAGES/SIemM/rdd.a
/usr/lib/locale/EN_US.ASCII/LC_MESSAGES/SIemM/rdd.d
/usr/lib/locale/EN_US.ASCII/LC_MESSAGES/SIemM/rdd.n
/usr/lib/locale/EN_US.ASCII/LC_MESSAGES/SIemM/rdd.r
/usr/lib/locale/EN_US.ASCII/LC_MESSAGES/SIemM/rdd.t
```

SEE ALSO

`getemm(1M)` .

Fujitsu Siemens Computers Add-on Reference Manual

rdswitch(8)

NAME

`rdswitch -` switch program for RAID controllers and LUNs of typePXRC, PXRE and EMC-SYMMETRIX

SYNOPSIS

Format 1

`rdswitch [ –r ] [ device ]`

Format 2

`rdswitch [ –r | –l ] [ device ]`

DESCRIPTION

The `rdswitch` command (no options) switches one or all LUNs from the current RAID controller to the specified partner controllers.

The `autoconf` node of the specified controller is also updated in the case of PXRC and PXRE systems, but not the node of the previous controller, i.e. the detached LUNs are still visible here. This node can be updated with the `–r` option.

In the case of EMC systems, all LUNs are permanently visible on both controllers in the `autoconf` tree; moreover, a LUN can be accessed by default from both controllers (status "Write Protect OFF", see the `–l` option).

The `rdswitch` command (no options) sets the status on the specified controller to "Write Protect OFF" and on the partner controller to "Write Protect ON" for one or all LUNs.

The `autoconf` node of the specified controller can be updated with the `–r` option. Since all LUNs are permanently visible on both controllers in the `autoconf` tree in the case of EMC systems, the `–r` option is only useful here in order to register new or obsolete LUNs in the `autoconf` tree.

The `–l` option only works on EMC systems; it returns "Write Protect" status for one or all LUNs of the specified controller ("Write Protect OFF" or "Write Protect ON") and consequently the up-to-date assignment of the LUNs to the controllers that are not visible from the `autoconf` tree.

OPTIONS

No option specified :

The specified LUN or all LUNs are switched to the specified controller and the `autoconf` node of this controller is updated.

Examples:

```
# rdswitch ios0/sraid160l2
```

Switches LUN 2 to the `sraid160` controller.

```
# rdswitch ios0/sraid160
```

Switches all LUNs to the `sraid160` controller.

`–r` The `autoconf` node of the specified LUN or specified controller is updated.

Examples:

```
# rdswitch –r ios0/sraid160l2# rdswitch –r ios0/sraid160
```

`–l` Only on EMC systems, otherwise the end status is 1.

The current "Write Protect" status is output for the specified LUN or for all LUNs of the specified controller; this status returns the assignment of the LUN to one or both controllers.

Examples:

```
# rdswitch –l ios0/sraid160l2
```

returns:

```
/dev/ios0/rsraid160l2  Write Protect OFF
```

```
# rdswitch –l ios0/sraid160
```

returns:

```
/dev/ios0/rsraid160l0  Write Protect OFF/dev/ios0/rsraid160l1  Write Protect
  ON/dev/ios0/rsraid160l2  Write Protect OFF/dev/ios0/rsraid160l3  Write Pro
tect  ON/dev/ios0/rsraid160l4  Write Protect OFF/dev/ios0/rsraid160l5  Write
 Protect  ON/dev/ios0/rsraid160l6  Write Protect OFF/dev/ios0/rsraid160l7  W
rite Protect  ON
```

EXIT STATUS

`0` Normal termination, function successful.

`1` Function unsuccessful or not available for this RAID system.

`2` Parameter errors.

FILES

`/dev/autoconf`
: Among other information, the `autoconf` tree contains the available RAID configurations that can be updated with `rdswitch –r`.

SEE ALSO

`autoconf`(8), `hszterm`(8), `reinitdev`(8).

Fujitsu Siemens Computers Add-on Reference Manual

rdup(8)

NAME

rdup - establishing the mount capability of RAID systems (PXRC,PXRE, EMC-SYMMETRIX) when booting

SYNOPSIS

rdup [init | start | stop]

DESCRIPTION

rdup is a startup script used to establish the operability (mount capability) of RAID systems (PXRC, PXRE, EMC-SYMMETRIX) at boot time.

Since it generally takes longer to boot RAID systems (particularly EMC-SYMMETRIX) than to boot the host, these RAID systems must be operational for the purpose of synchronizing the mount activities (Definition: a RAID system is operational if it is entered in the autoconf tree).

To safeguard this synchronization, the RAID startup check (rdup start) uses an RC2 S01rdup script to cyclically compare the RAID controllers in the autoconf tree with the target configuration predefined and expected in a RAID configuration file (/etc/rdstartup).

A reprobe [see reinitdev(8)] is carried out on the affected SCSI port (sport) for each outstanding RAID controller. The RAID startup check is terminated when all RAID controllers have been found in the autoconf tree; the DRAID draidd(1M) daemon, assuming it is active, is restarted to take account of the current configuration.

If all controllers are not found in the autoconf tree, the RAID startup check ends after 60 minutes with a timeout; moreover, the system administrator can abort the RAID startup check at any time with <CR> plus confirmation. A switch is made to single user mode in both cases to give the system administrator the opportunity to make corrections. Any RAID controllers still outstanding are logged in /dev/osm.

The RAID /etc/rdstartup configuration file must be created by the system administrator using the rdup init command from an up-to-date autoconf tree. If there is no configuration file available, the RAID startup check assumes that there are no RAIDs configured on this system.

Notes:

- Since S01rdup is started after S01MOUNT0FSYS, Level-0 file systems must not be mounted on RAID systems!
- You need root rights to use rdup.

OPTIONS

init The system administrator can/must use the rdup init command to create the /etc/rdstartup RAID configuration file. The existing autoconf tree is used here. At boot time (rdup start), this configuration file is compared cyclically with the respective autoconf status.

The configuration file contains the RAID ports (sports) with the associated controllers, for example:

```
# cat /etc/rdstartup
sport06 sraid060        # EMC-SYMMETRIX
sport06 sraid061        # PXRE-F
sport07 sraid070        # PXRC
.....
```

If no RAID systems are found in the autoconf tree, no configuration file will be created.

init always terminates with exit status 0.

Important:

- rdup init must be invoked manually by the system administrator both after the first installation and after every configuration change in RAID systems.

- The system administrator must ensure that all RAID systems are entered in the `autoconf` tree at this time.
- The system administrator can edit the configuration file for the purpose of selecting the relevant RAID systems for monitoring.

`start`

`rdup start` is executed as an RC2 `S01rdup` script (RAID startup check) when the system boots.

`rdup start` cyclically compares (every 30 seconds) the RAID controllers available (= operational) in the current `autoconf` tree with the RAID controllers entered in the configuration file. If no configuration file exists, the exit status 0 is returned immediately.

Provided there are still RAID controllers missing in the `autoconf` tree, a reprobe [see `reinitdev(8)`] will be carried out on the associated ports (`sports`).

As soon as the `autoconf` configuration complies with the configuration file, the RAID startup check terminates with exit status 0; moreover, the DRAID `draidd(1M)` daemon is restarted, assuming it is active, to take account of the current RAID status.

Termination with a timeout:

If a controller is not found, a timeout occurs after 60 minutes. The user can also abort the RAID startup check at any time by hitting <CR> (plus positive acknowledgement). In both cases, exit status 100 forces a switch to single user mode to give the system administrator the opportunity to make corrections; the outstanding RAID systems are logged in `/dev/osm`.

`stop` Dummy option with exit status 0, to comply with the RC script conventions.

FILES

`/dev/autoconf`

This device node contains the currently operational RAID systems, for example.

`/etc/rdstartup`

RAID configuration created with `rdup init`; it contains the RAID configuration that is compared at boot time (`rdup start`) with the current `autoconf` status. The configuration file contains the RAID ports (`sports`) with the associated controllers, e.g.:

```
# cat /etc/rdstartup
sport06 sraid060        # EMC-SYMMETRIX
sport06 sraid061        # PXRE-F
sport07 sraid070        # PXRC
....
```

Notes:

- In theory, you can connect various RAID systems to a `sport` port; this is taken into account when the configuration file is set up.
- Using an editor, users can select the RAID systems (to be monitored) that are relevant for them.

`/dev/osm`

When the RAID startup check is aborted by a timeout or by the user, the overdue RAID systems are logged in `/dev/osm` for posterity.

An entry is also logged in `/dev/osm` if there are more RAID systems in the `autoconf` tree than are entered in the configuration file; this may happen if the configuration file is not up to date or has been modified.

EXIT STATUS

`0` Normal termination:

- All RAID systems are to be booted according to the configuration file or
- no `/etc/rdstartup` configuration file is available, i.e. no RAID systems are connected.
- `rdup init` is always accompanied by exit status 0; if a configuration file has not been created, no

RAID systems are configured or exist in the `autoconf` tree.

`100` Abnormal termination (exit status 100 forces single user mode) with a

- timeout after 60 minutes or
- abort of the startup check by a user prompt.

The RAID systems that are still outstanding at this time are logged in `/dev/osm`.

SEE ALSO

`draidd`(1M), `autoconf`(8), `reinitdev`(8).