ChorusOS man pages section 4CC: Files

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part No: 806-3337
December 10, 1999

Contents

PREFACE

Overview

A man page is provided for both the naive user, and sophisticated user who is familiar with the ChorusOS™ operating system and is in need of on-line information. A man page is intended to answer concisely the question “What does it do?” The man pages in general comprise a reference manual. They are not intended to be a tutorial.

The following is a list of sections in the ChorusOS man pages and the information it references:

- *Section 1CC: User Utilities; Host and Target Utilities*
- *Section 1M: System Management Utilities*
- *Section 2DL: System Calls; Data Link Services*
- *Section 2K: System Calls; Kernel Services*
- *Section 2MON: System Calls; Monitoring Services*
- *Section 2POSIX: System Calls; POSIX System Calls*
- *Section 2RESTART: System Calls; Hot Restart and Persistent Memory*
- *Section 2SEG: System Calls; Virtual Memory Segment Services*
- *Section 3FTPD: Libraries; FTP Daemon*
- *Section 3M: Libraries; Mathematical Libraries*
- *Section 3POSIX: Libraries; POSIX Library Functions*
- *Section 3RPC: Libraries; RPC Services*
- *Section 3STDC: Libraries; Standard C Library Functions*
- *Section 3TELD: Libraries; Telnet Services*
- *Section 4CC: Files*

- *Section 5FEA: ChorusOS Features and APIs*
- *Section 7P: Protocols*
- *Section 7S: Services*
- *Section 9DDI: Device Driver Interfaces*
- *Section 9DKI: Driver to Kernel Interface*
- *Section 9DRV: Driver Implementations*

ChorusOS man pages are grouped in Reference Manuals, with one reference manual per section.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the `intro` pages for more information and detail about each section, and `man`(1) for more information about man pages in general.

NAME — This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS — This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full pathname is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ] — The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.

. . . — Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, ‘ "`filename` . . ." .

| — Separator. Only one of the arguments separated by this character can be specified at time.

{ } — Braces. The options and / or arguments enclosed within braces are

	interdependent, such that everything enclosed must be treated as a unit.
FEATURES	This section provides the list of features which offer an interface. An API may be associated with one or more system features. The interface will be available if one of the associated features has been configured.
DESCRIPTION	This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES.. Interactive commands, subcommands, requests, macros, functions and such, are described under USAGE.
OPTIONS	This lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.
OPERANDS	This section lists the command operands and describes how they affect the actions of the command.
OUTPUT	This section describes the output - standard output, standard error, or output files - generated by the command.
RETURN VALUES	If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.
ERRORS	On failure, most functions place an error code in the global variable `errno` indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.

USAGE	This section is provided as a guidance on use. This section lists special rules, features and commands that require in-depth explanations. The subsections listed below are used to explain built-in functionality: Commands Modifiers Variables Expressions Input Grammar
EXAMPLES	This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%` or if the user must be superuser, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS and USAGE sections.
ENVIRONMENT VARIABLES	This section lists any environment variables that the command or function affects, followed by a brief description of the effect.
EXIT STATUS	This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion and values other than zero for various error conditions.
FILES	This section lists all filenames referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.
SEE ALSO	This section lists references to other man pages, in-house documentation and outside publications.
DIAGNOSTICS	This section lists diagnostic messages with a brief explanation of the condition causing the error.
WARNINGS	This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.
NOTES	This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.

BUGS

This section describes known bugs and wherever possible, suggests workarounds.

CHAPTER

Files

10

NAME

Intro – file formats used or read by various programs

DESCRIPTION

This section describes the formats of files used by various programs.

LIST OF FILE FORMATS

Name:	Appears on Page:	Description:
disktab	disktab(4CC)	disk description file
exports	exports(4CC)	remote mount points for NFS mount requests
fstab	fstab(4CC)	static information about the file system
hosts	hosts(4CC)	host name data base
netgroup	netgroup(4CC)	list of network groups
protocols	protocols(4CC)	protocol name database
resolv.conf	resolv.conf(4CC)	configuration file for DOMAIN name system resolver
security	security(4CC)	user security file
services	services(4CC)	Internet services and aliases
tzfile	tzfile(4CC)	time zone infomation

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

NAME

dhclient.cf – DHCP client configuration file

DESCRIPTION

The `dhclient.cf` file contains configuration information for `dhclient`, the Internet Software Consortium DHCP Client.

The `dhclient.cf` file is a free-form ASCII text file. It is parsed by the recursive-descent parser built into `dhclient`(1M). The file may contain extra tabs and newlines for formatting purposes. Keywords in the file are case-insensitive. Comments may be placed anywhere within the file (except within quotes). Comments begin with the # character and end at the end of the line.

The `dhclient.cf` file can be used to configure the behavior of the client in a wide variety of ways: protocol timing, information requested from the server, information required of the server, defaults to use if the server does not provide certain information, values with which to override information provided by the server, or values to prepend or append to information provided by the server. The configuration file can also be preinitialized with addresses to use on networks that dov not have DHCP servers.

WARNINGS

Many options are defined for DHCP. The implementation of `dhclient` in this release takes only the `subnet-mask` option into account. Other options are silently ignored.

PROTOCOL TIMING

You do not need to configure the timing behavior of the client. If no timing configuration is provided by the user, a fairly reasonable timing behaviour will be used by default — one which results in fairly timely updates without placing an inordinate load on the server.

The following statements can be used to adjust the timing behaviour of the DHCP client if required, however:

The `timeout` statement

timeout *time* ;

The `timeout` statement determines the amount of time that must pass between the time that the client begins to try to determine its address and the time that it decides that it's not going to be able to contact a server. By default, this timeout is sixty seconds. After the timeout has passed, if there are any static leases defined in the configuration, the client will loop through these leases attempting to validate them, and if it finds one that appears to be valid, it will use that lease's address. If there are no valid static leases, the client will restart the protocol after the defined retry interval.

The `retry` statement

retry *time* ;

The `retry` statement determines the time that must pass after the client has determined that there is no DHCP server present before it tries again to contact a DHCP server. By default, this is five minutes.

The `select-timeout` statement

select-timeout *time* ;

It is possible (some might say desirable) for there to be more than one DHCP server serving any given network. In this case, it is possible that a client may be sent more than one offer in response to its initial lease discovery message. It may be that one of these offers is preferable to the other (e.g., one offer may have the address the client previously used, and the other may not).

The `select-timeout` is the time after the client sends its first lease discovery request at which it stops waiting for offers from servers, assuming that it has received at least one such offer. If no offers have been received by the time the select-timeout has expired, the client will accept the first offer that arrives.

By default, the select-timeout is zero seconds — that is, the client will take the first offer it sees.

The `reboot` statement

reboot *time* ;

When the client is restarted, it first tries to reacquire the last address it had. This is called the `INIT-REBOOT` state. If it is still attached to the same network it was attached to when it last ran, this is the quickest way to get started. The reboot statement sets the time that must elapse after the client first tries to reacquire its old address before it gives up and tries to discover a new address. By default, the `reboot` timeout is ten seconds.

The `backoff-cutoff` statement

backoff-cutoff *time* ;

The client uses an exponential backoff algorithm with some randomness, so that if many clients try to configure themselves at the same time, they will not make their requests in lockstep. The `backoff-cutoff` statement determines the maximum amount of time that the client is allowed to back off. It defaults to two minutes.

The `initial-interval` statement

initial-interval *time* ;

The `initial-interval` statement sets the amount of time between the first attempt to reach a server and the second attempt to reach a server. Each time a message is sent, the interval between messages is incremented by twice the current interval multiplied by a random number between zero and one. If it is greater than the backoff-cutoff amount, it is set to that amount. It defaults to ten seconds.

LEASE REQUIREMENTS AND REQUESTS

The DHCP protocol allows the client to request that the server send it specific information, and not send it other information that it is not prepared to accept. The protocol also allows the client to reject offers from servers if they don’t contain information the client needs, or if the information provided is not satisfactory.

There is a variety of data contained in offers that DHCP servers send to DHCP clients. The data that can be specifically requested is what are called DHCP Options. DHCP Options are defined in `dhcp-options`.

The `request` statement

request [*option*…] ;

The `request` statement causes the client to request that any server responding to the client send the client its values for the specified options. Only the option names should be specified in the request statement — not option parameters.

The `require` statement

require [*option*…] ;

The `require` statement lists options that must be sent in order for an offer to be accepted. Offers that do not contain all the listed options will be ignored.

The `send` statement

send { [*option declaration*…] }

The `send` statement causes the client to send the specified options to the server with the specified values. These are full option declarations as described in `dhcp-options`. Options that are always sent in the DHCP protocol should not be specified here, except that the client can specify a requested-lease-time option other than the default requested lease time, which is two hours. The other obvious use for this statement is to send information to the server that will allow it to differentiate between this client and other clients or kinds of clients.

OPTION MODIFIERS

In some cases, a client may receive option data from the server which is not really appropriate for that client, or may not receive information that it needs, and for which a useful default value exists. It may also receive information which is useful, but which needs to be supplemented with local information. To handle these needs, several option modifiers are available.

The `default` statement

default { *option declaration*… }

If for some set of options the client should use the value supplied by the server, but needs to use some default value if no value was supplied by the server, these values can be defined in the `default` statement.

The `supercede` statement

supercede { [*option declaration*…] }

If for some set of options the client should always use its own value rather than any value supplied by the server, these values can be defined in the `supersede` statement.

The `prepend` statement

prepend { *option declaration*… }

If for some set of options the client should first use a value it supplies, and then use the values supplied by the server, if any. These values can be defined in the `prepend` statement. The `prepend` statement can only be used for options which allow more than one value to be given.

The append **statement**

append { *option declaration*… }

If for some set of options the client should first use a value supplied by the server, and then use a value it supplies, if any. These values can be defined in the `append` statement. The `append` statement can only be used for options which allow more than one value to be given.

LEASE DECLARATIONS

The `lease` **statement**

lease { *lease-declaration*… }

The DHCP client may decide after some period of time (see *PROTOCOL TIMING*) that it is not going to succeed in contacting a server. It is possible to define one or more *fixed* leases in the client configuration file for networks where there is no DHCP or BOOTP service, so that the client can still automatically configure its address. This is done with the lease statement.

A `lease` statement consists of the lease keyword, followed by a left curly brace, followed by one or more lease declaration statements, followed by a right curly brace. The following lease declarations are possible:

bootp;
: The `bootp` statement is used to indicate that the lease was acquired using the BOOTP protocol rather than the DHCP protocol. It is never necessary to specify this in the client configuration file. The client uses this syntax in its lease database file.

interface “*string*”;
: The `interface` lease statement is used to indicate the interface on which the lease is valid. If set, this lease will only be tried on a particular interface. When the client receives a lease from a server, it always records the interface number on which it received that lease. If predefined leases are specified in the `dhclient.cf` file, the interface should also be specified, although this is not required.

fixed-address *IP_address*;
: The `fixed-address` statement is used to set the ip address of a particular lease. This is required for all lease statements. The IP address must be specified as a dotted quad (for example, 12.34.56.78).

filename “*string*”;
: The `filename` statement specifies the name of the boot filename to use. This is not used by the standard client configuration script, but is included for completeness.

server-name “*string*”;
: The `server-name` statement specifies the name of the boot server name to use. This is also not used by the standard client configuration script.

option *option-declaration*;
The `option` statement is used to specify the value of an option supplied by the server, or, in the case of predefined leases declared in `dhclient.cf`, the value that the user wishes the client configuration script to use if the predefined lease is used.

medium "*media setup*";
The `medium` statement can be used on systems where network interfaces cannot automatically determine the type of network to which they are connected. The media setup string is a system-dependent parameter which is passed to the DHCP client configuration script when initializing the interface. On Unix and Unix-like systems, the argument is passed on the `ifconfig` command line when configuring the interface.

The DHCP client automatically declares this parameter if it used a media type (see the `media` statement) when configuring the interface in order to obtain a lease. This statement should be used in predefined leases only if the network interface requires media type configuration.

renew *date*;
rebind *date*;
expire *date*;
The `renew` statement defines the time at which the DHCP client should begin trying to contact its server to renew a lease that it is using. The rebind statement defines the time at which the DHCP client should begin to try to contact any DHCP server in order to renew its lease. The expire statement defines the time at which the DHCP client must stop using a lease if it has not been able to contact a server in order to renew it.

These declarations are automatically set in leases acquired by the DHCP client, but must also be configured in predefined leases — a predefined lease whose expiry time has passed will not be used by the DHCP client.

Dates are specified as follows:
weekday
year / *month* / *day*
hour:*minute*:*second*

The weekday is present to make it easy for a human to tell when a lease expires — it's specified as a number from zero to six, with zero being Sunday. When declaring a predefined lease, it can always be specified as zero. The year is specified with the century, so it should generally be four digits except for really long leases. The month is specified as a number starting with 1 for January. The day of the month is likewise specified starting with 1. The hour is a number between 0 and 23, the minute a number between 0 and 69, and the second also a number between 0 and 69.

ALIAS DECLARATIONS

alias { *declaration*… }

Some DHCP clients running TCP/IP roaming protocols may require that in addition to the lease they may acquire via DHCP, their interface also be configured with a predefined IP alias so that they can have a permanent IP address even while roaming. The Internet Software Consortium DHCP client doesn't support roaming with fixed addresses directly, but in order to facilitate such experimentation, the DHCP client can be set up to configure an IP alias using the alias declaration.

The `alias` declaration resembles a `lease` declaration, except that options other than the subnet-mask option are ignored by the standard client configuration script, and expiry times are ignored. A typical alias declaration includes an interface declaration, a fixed-address declaration for the IP alias address, and a subnet-mask option declaration. A `medium` statement should never be included in an alias declaration.

OTHER DECLARATIONS

reject *IP-address*;

The `reject` statement causes the DHCP client to reject offers from servers who use the specified address as a server identifier. This can be used to avoid being configured by rogue or misconfigured DHCP servers, although it should be a last resort — better to track down the bad DHCP server and fix it.

interface "*name*" { *declarations ...* }

A client with more than one network interface may require different behaviour depending on which interface is being configured. All timing parameters and declarations other than lease and alias declarations can be enclosed in an `interface` declaration, and those parameters will then be used only for the interface that matches the specified name. Interfaces for which there is no interface declaration will use the parameters declared outside of any interface declaration, or the default settings.

media "*media setup*" [, "*media setup*", ...];

The `media` statement defines one or more media configuration parameters which may be tried while attempting to acquire an IP address. The DHCP client will cycle through each media setup string on the list, configuring the interface using that setup and attempting to boot, and then trying the next one. This can be used for network interfaces which aren't capable of sensing the media type unaided — whichever media type succeeds in getting a request to the server and hearing the reply is probably right (no guarantees).

The media setup is only used for the initial phase of address acquisition (the DHCPDISCOVER and DHCPOFFER packtes). Once an address has been acquired, the DHCP client will record it in its lease database and will record

the media type used to acquire the address. Whenever the client tries to renew the lease, it will use that same media type. The lease must expire before the client will go back to cycling through media types.

EXAMPLES

The following configuration file is used on a laptop running NetBSD 1.3. The laptop has an IP alias of 192.5.5.213, and has one interface, ep0 (a 3com 3C589C). Booting intervals have been shortened somewhat from the default, because the client is known to spend most of its time on networks with little DHCP activity. The laptop does roam to multiple networks.

```
timeout 60; retry 60; reboot 10; select-timeout 5;
initial-interval 2; reject 192.33.137.209;

interface "ep0" {
    send host-name "andare.fugue.com"; send
    dhcp-client-identifier 1:0:a0:24:ab:fb:9c; send
    dhcp-lease-time 3600; supersede domain-name "fugue.com
    rc.vix.com home.vix.com"; prepend domain-name-servers
    127.0.0.1; request subnet-mask, broadcast-address,
    time-offset, routers,
     domain-name, domain-name-servers, host-name;
    require subnet-mask, domain-name-servers;
    media "media 10baseT/UTP", "media 10base2/BNC";
}

alias {
  interface "ep0"; fixed-address 192.5.5.213; option subnet-mask
  255.255.255.255;
}
```

This is a very complicated `dhclient.cf` file — in general, yours should be much simpler. In many cases, you need no `dhclient.cf` file — the defaults are usually fine.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`dhclient`(1M), `dhcp-options`(4CC), `bpf`(7S)

RFC2132, *RFC2131*

AUTHOR

`dhclient` was written by Ted Lemon <mellon@vix.com> under a contract with Vixie Labs. Funding for this project was provided by the Internet Software Corporation. Information about the Internet Software Consortium can be found at `http://www.isc.org/isc`.

NAME

dhcp–options – Dynamic Host Configuration Protocol options

DESCRIPTION

The Dynamic Host Configuration Protocol allows the client to receive options from the DHCP server describing the network configuration and various services that are available on the network. When configuring `dhclient`, options must often be declared. The syntax for declaring options, and the names and formats of the options that can be declared, are documented here.

WARNINGS

Many options are defined for DHCP. The implementation of `dhclient` in this release takes only the `subnet-mask` option into account. Other options are silently ignored.

OPTION STATEMENTS

DHCP option statements always start with the option keyword, followed by an option name, followed by option data. The option names and data formats are described below. It is not necessary to exhaustively specify all DHCP options — only those options which are needed by clients must be specified.

Option data comes in a variety of formats, as defined below:

The *ip-address* data type can be entered either as an explicit IP address as a domain name. When entering a domain name, be sure that that domain name resolves to a single IP address.

The *int32* data type specifies a signed 32-bit integer. The *uint32* data type specifies an unsigned 32-bit integer. The *int16* and *uint16* data types specify signed and unsigned 16-bit integers. The *int8* and *uint8* data types specify signed and unsigned 8-bit integers. Unsigned 8-bit integers are also sometimes referred to as *octets*.

The string data type specifies an NVT ASCII string, which must be enclosed in double quotes — for example, to specify a domain-name option, the syntax would be

```
option domain-name "sun.com";
```

The flag data type specifies a boolean value. Booleans can be either true or false (or on or off, if that makes more sense to you).

The data-string data type specifies either an NVT ASCII string enclosed in double quotes, or a series of octets specified in hexadecimal, separated by colons. For example:

```
option client-identifier "CLIENT-FOO";
```

or

```
option client-identifier 43:4c:49:45:54:2d:46:4f:4f;
```

The documentation for the various options mentioned below is taken from the latest IETF draft document on DHCP options. Options which are not listed by name may be defined by the name option-nnn, where nnn is the decimal number

of the option code. These options may be followed either by a string, enclosed in quotes, or by a series of octets, expressed as two-digit hexadecimal numbers separated by colons. For example:

```
option option-133 "my-option-133-text";

option option-129 1:54:c9:2b:47;
```

Because `dhclient` does not know the format of these undefined option codes, no checking is done to ensure the correctness of the entered data.

The standard options are:

`option subnet-mask` *ip-address*`;`

The subnet mask option specifies the client's subnet mask as described in *RFC 950*. If no subnet mask option is provided anywhere in scope, as a last resort dhclient will use the subnet mask from the subnet declaration for the network on which an address is being assigned. However, any subnet-mask option declaration that is in scope for the address being assigned will override the subnet mask specified in the subnet declaration.

`option time-offset` *int32*`;`

The time-offset option specifies the offset of the client's subnet in seconds from Coordinated Universal Time (UTC).

`option routers` *ip-address* `[,` *ip-address* `... ];`

The routers option specifies a list of IP addresses for routers on the client's subnet. Routers should be listed in order of preference.

`option time-servers` *ip-address* `[,` *ip-address* `... ];`

The time-server option specifies a list of *RFC 868* time servers available to the client. Servers should be listed in order of preference.

`option ien116-name-servers` *ip-address* `[,` *ip-address* `... ];`

The ien116-name-servers option specifies a list of IEN 116 name servers available to the client. Servers should be listed in order of preference.

`option domain-name-servers` *ip-address* `[,` *ip-address* `... ];`

The domain-name-servers option specifies a list of Domain Name System (STD 13, *RFC 1035*) name servers available to the client. Servers should be listed in order of preference.

`option log-servers` *ip-address* `[,` *ip-address* `... ];`

The log-server option specifies a list of MIT-LCS UDP log servers available to the client. Servers should be listed in order of preference.

`option cookie-servers` *ip-address* `[,` *ip-address* `... ];`

The cookie server option specifies a list of *RFC 865* cookie servers available to the client. Servers should be listed in order of preference.

`option lpr-servers` *ip-address* `[,` *ip-address* `... ];`

The LPR server option specifies a list of *RFC 1179* line printer servers available to the client. Servers should be listed in order of preference.

`option impress-servers` *ip-address* `[,` *ip-address* `... ];`

The impress-server option specifies a list of Imagen Impress servers available to the client. Servers should be listed in order of preference.

`option resource-location-servers` *ip-address* `[,` *ip-address* `... ];`

This option specifies a list of *RFC 887* Resource Location servers available to the client. Servers should be listed in order of preference.

`option host-name` *string*`;`

This option specifies the name of the client. The name may or may not be qualified with the local domain name (it is preferable to use the domain-name option to specify the domain name). See *RFC 1035* for character set restrictions.

`option boot-size` *uint16*`;`

This option specifies the length in 512-octet blocks of the default boot image for the client.

`option merit-dump` *string*`;`

This option specifies the path-name of a file to which the client's core image should be dumped in the event the client crashes. The path is formatted as a character string consisting of characters from the NVT ASCII character set.

`option domain-name` *string*`;`

This option specifies the domain name that client should use when resolving hostnames via the Domain Name System.

`option swap-server` *ip-address*`;`

This specifies the IP address of the client's swap server.

`option root-path` *string*`;`

This option specifies the path-name that contains the client's root disk. The path is formatted as a character string consisting of characters from the NVT ASCII character set.

`option ip-forwarding` *flag*`;`

This option specifies whether the client should configure its IP layer for packet forwarding. A value of 0 means disable IP forwarding, and a value of 1 means enable IP forwarding.

`option non-local-source-routing` *flag*`;`

This option specifies whether the client should configure its IP layer to allow forwarding of datagrams with non-local source routes (see Section 3.3.5 of [4] for a discussion of this topic). A value of 0 means disallow forwarding of such datagrams, and a value of 1 means allow forwarding.

`option policy-filter` *ip-address ip-address* `[,` *ip-address ip-address* `... ];`

This option specifies policy filters for non-local source routing. The filters consist of a list of IP addresses and masks which specify destination / mask pairs with which to filter incoming source routes.

Any source routed datagram whose next-hop address does not match one of the filters should be discarded by the client.

See STD 3 (*RFC1122*) for further information.

`option max-dgram-reassembly` *uint16*`;`

This option specifies the maximum size datagram that the client should be prepared to reassemble. The minimum value legal value is `576`.

`option default-ip-ttl` *uint8*`;`

This option specifies the default time-to-live that the client should use on outgoing datagrams.

`option path-mtu-aging-timeout` *uint32*`;`

This option specifies the timeout (in seconds) to use when aging Path MTU values discovered by the mechanism defined in *RFC 1191*.

`option path-mtu-plateau-table` *uint16* `[,` *uint16* `... ];`

This option specifies a table of MTU sizes to use when performing Path MTU Discovery as defined in *RFC 1191*. Thetable is formatted as a list of 16-bit unsigned integers, ordered from smallest to largest. The minimum MTU value cannot be smaller than 68.

`option interface-mtu` *uint16*`;`

This option specifies the MTU to use on this interface. The minimum legal value for the MTU is 68.

`option all-subnets-local` *flag*`;`

This option specifies whether or not the client may assume that all subnets of the IP network to which the client is connected use the same MTU as the subnet of that network to which the client is directly connected. A value of 1 indicates that all subnets share the same MTU. A value of 0 means that the client should assume that some subnets of the directly connected network may have smaller MTUs.

`option broadcast-address` *ip-address*`;`

This option specifies the broadcast address in use on the client's subnet. Legal values for broadcast addresses are specified in section 3.2.1.3 of STD 3 (*RFC 1122*).

`option perform-mask-discovery` *flag*`;`

This option specifies whether or not the client should perform subnet mask discovery using ICMP. A value of 0 indicates that the client should not perform mask discovery. A value of 1 means that the client should perform mask discovery.

`option mask-supplier` *flag*`;`

This option specifies whether or not the client should respond to subnet mask requests using ICMP. A value of 0 indicates that the client should not respond. A value of 1 means that the client should respond.

`option router-discovery` *flag*`;`

This option specifies whether or not the client should solicit routers using the Router Discovery mechanism defined in *RFC 1256*. A value of 0 indicates that the client should not perform router discovery. A value of 1 means that the client should perform router discovery.

`option router-solicitation-address` *ip-address*`;`

This option specifies the address to which the client should transmit router solicitation requests.

`option static-routes` *ip-address ip-address* `[,` *ip-address ip-address* `... ];`

This option specifies a list of static routes that the client should install in its routing cache. If multiple routes to the same destination are specified, they are listed in descending order of priority.

The routes consist of a list of IP address pairs. The first address is the destination address, and the second address is the router for the destination.

The default route (0.0.0.0) is an illegal destination for a static route. To specify the default route, use the routers option.

`option trailer-encapsulation` *flag*`;`

This option specifies whether or not the client should negotiate the use of trailers (RFC 893 [14]) when using the ARP protocol. A value of 0 indicates that the client should not attempt to use trailers. A value of 1 means that the client should attempt to use trailers.

`option arp-cache-timeout` *uint32*`;`

This option specifies the timeout in seconds for ARP cache entries.

`option ieee802-3-encapsulation` *flag*`;`

This option specifies whether or not the client should use Ethernet Version 2 (RFC 894) or IEEE 802.3 (RFC 1042) encapsulation if the interface is an Ethernet. A value of 0 indicates that the client should use RFC 894 encapsulation. A value of 1 means that the client should use RFC 1042 encapsulation.

`option default-tcp-ttl` *uint8*`;`

This option specifies the default TTL that the client should use when sending TCP segments. The minimum value is 1.

`option tcp-keepalive-interval` *uint32*`;`

This option specifies the interval (in seconds) that the client TCP should wait before sending a keepalive message on a TCP connection. The time is specified as a 32-bit unsigned integer. A value of zero indicates that the client should not generate keepalive messages on connections unless specifically requested by an application.

`option tcp-keepalive-garbage` *flag*`;`

This option specifies the whether or not the client should send TCP keepalive messages with a octet of garbage for compatibility with older implementations. A value of 0 indicates that a garbage octet should not be sent. A value of 1 indicates that a garbage octet should be sent.

`option nis-domain` *string*`;`

This option specifies the name of the client's NIS (Sun Network Information Services) domain. The domain is formatted as a character string consisting of characters from the NVT ASCII character set.

`option nis-servers` *ip-address* `[,` *ip-address* `... ];`

This option specifies a list of IP addresses indicating NIS servers available to the client. Servers should be listed in order of preference.

`option ntp-servers` *ip-address* `[,` *ip-address* `... ];`

This option specifies a list of IP addresses indicating NTP (*RFC 1035*) servers available to the client. Servers should be listed in order of preference.

`option netbios-name-servers` *ip-address* `[,` *ip-address* `... ];`

The NetBIOS name server (NBNS) option specifies a list of RFC 1001/1002 NBNS name servers listed in order of preference.

`option netbios-dd-server` *ip-address* `[,` *ip-address* `... ];`

The NetBIOS datagram distribution server (NBDD) option specifies a list of RFC 1001/1002 NBDD servers listed in order of preference.

`option netbios-node-type` *uint8*`;`

The NetBIOS node type option allows NetBIOS over TCP/IP clients which are configurable to be configured as described in RFC 1001/1002. The value is specified as a single octet which identifies the client type. A value of 1 corresponds to a NetBIOS B-node; a value of 2 corresponds to a P-node; a value of 4 corresponds to an M-node; a value of 8 corresponds to an H-node.

`option netbios-scope` *string*`;`

The NetBIOS scope option specifies the NetBIOS over TCP/IP scope parameter for the client as specified in RFC 1001/1002. See RFC1001, RFC1002, and RFC1035 for character-set restrictions.

`option font-servers` *ip-address* `[,` *ip-address* `... ];`

This option specifies a list of X Window System Font servers available to the client. Servers should be listed in order of preference.

`option x-display-manager` *ip-address* `[,` *ip-address* `... ];`

This option specifies a list of systems that are running the X Window System Display Manager and are available to the client. Addresses should be listed in order of preference.

`option dhcp-client-identifier` *data-string*`;`

This option can be used to specify the a DHCP client identifier in a host declaration, so that dhcpd can find the host record by matching against the client identifier.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`dhclient`(1M), `dhclient.5cf`(4CC), `bpf`(7S)

AUTHOR

`dhclient` was written by Ted Lemon <mellon@vix.com> under a contract with Vixie Labs. Funding for this project was provided by the Internet Software Corporation. Information about the Internet Software Consortium can be found at `http://www.isc.org/isc`.

This client was substantially modified and enhanced by Elliot Poger for use on Linux while he was working on the MosquitoNet project at Stanford.

The current version owes much to Elliot's Linux enhancements, but was substantially reorganized and partially rewritten by Ted Lemon so as to use the same networking framework that the Internet Software Consortium DHCP server uses. Much system-specific configuration code was moved into a shell script so that as support for more operating systems is added, it will not be necessary to port and maintain system-specific configuration code to these operating systems — instead, the shell script can invoke the native tools to accomplish the same purpose.

NAME

disklabel – disk pack label

DESCRIPTION

Each disk or disk pack on a system can contain a disk label which provides detailed information about the geometry of the disk and the partitions into which the disk is divided. It should be initialized when the disk is formatted, and may be changed later using the `disklabel`(1M) program. This information is used by the system disk driver and by the bootstrap program to determine how to program the drive and where to find the filesystems on the disk partitions. Additional information is used by the filesystem in order to use the disk most efficiently and to locate important filesystem information. The description of each partition contains an identifier for the partition type (for example, standard filesystem, swap area). The filesystem updates the in-core copy of the label if it contains incomplete information about the filesystem.

The label is located in sector number *LABELSECTOR* of the drive, usually sector 0 where it may be found without any information about the disk geometry. It is at an offset *LABELOFFSET* from the beginning of the sector, to allow room for the initial bootstrap. The disk sector containing the label is usually made read-only so that it is not accidentally overwritten by pack-to-pack copies or swap operations; the *DIOCWLABEL* `ioctl`(2POSIX) is done as needed by the `disklabel` program.

A copy of the in-core label for a disk can be obtained using *DIOCGDINFO ioctl;* this works with a file descriptor for a block or character (“raw”) device for any partition of the disk. The in-core copy of the label is set by *DIOCSDINFO ioctl.* The offset of a partition cannot generally be changed while it is open, nor can it be made smaller while it is open. One exception is that any change is allowed if no label was found on the disk, and the driver was only able to construct a skeletal label without partition information. Finally, the *DIOCWDINFO ioctl* operation sets the in-core label and then updates the on-disk label; there must be an existing label on the disk for this operation to succeed. Thus, the initial label for a disk or disk pack must be installed by writing to the raw disk. All of these operations are normally performed using *disklabel* .

```
/*
* Disk description table, see  disktab(4CC)

*/
#define DISKTAB         (/etc/disktab)
/* Each disk has a label which includes information about the
 * hardware disk geometry, filesystem partitions, and drive
 * specific information.  The label is in block 0 or 1, possibly
 * offset from the beginning to leave room for a bootstrap, etc.
 */

#ifndef LABELSECTOR
#define LABELSECTOR     0                   /* sector containing label */
#endif
```

```
#ifndef LABELOFFSET
#define LABELOFFSET     64                   /* offset of label in sector */
#endif

#define DISKMAGIC       ((u_long) 0x82564557) /* The disk magic number */
#ifndef MAXPARTITIONS
#define MAXPARTITIONS   8
#endif

#ifndef LOCORE
struct disklabel {
        u_long  d_magic;        /* the magic number */
        short   d_type;         /* drive type */
        short   d_subtype;      /* controller/d_type specific */
        char    d_typename[16]; /* type name, e.g. (eagle" */
        /*
        * d_packname contains the pack identifier and is returned when
        * the disklabel is read off the disk or in-core copy.
        * d_boot0 and d_boot1 are the (optional) names of the
        * primary (block 0) and secondary (block 1-15) bootstraps
        * as found in /usr/mdec.  These are returned when using
        * getdiskbyname(3)
        to retrieve the values from /etc/disktab.
        */
#if defined(KERNEL) || defined(STANDALONE)
        char    d_packname[16];         /* pack identifier */
#else
        union {
                char    un_d_packname[16];      /* pack identifier */
                struct {
                        char *un_d_boot0;       /* primary bootstrap name */
                        char *un_d_boot1;       /* secondary bootstrap name */
                } un_b;
        } d_un;

#define d_packname      d_un.un_d_packname
#define d_boot0         d_un.un_b.un_d_boot0
#define d_boot1         d_un.un_b.un_d_boot1
#endif  /* ! KERNEL or STANDALONE */

        /* disk geometry: */
        u_long  d_secsize;      /* # of bytes per sector */
        u_long  d_nsectors;     /* # of data sectors per track */
        u_long  d_ntracks;      /* # of tracks per cylinder */
        u_long  d_ncylinders;   /* # of data cylinders per unit */
        u_long  d_secpercyl;    /* # of data sectors per cylinder */
        u_long  d_secperunit;   /* # of data sectors per unit */
        /*
        * Spares (bad sector replacements) below
        * are not counted in d_nsectors or d_secpercyl.
        * Spare sectors are assumed to be physical sectors
        * which occupy space at the end of each track and/or cylinder.
        */
        u_short d_sparespertrack;       /* # of spare sectors per track */
        u_short d_sparespercyl; /* # of spare sectors per cylinder */
```

```
        /*
        * Alternate cylinders include maintenance, replacement,
        * configuration description areas, etc.
        */
        u_long  d_acylinders;   /* # of alt. cylinders per unit */

                /* hardware characteristics: */
        /*
        * d_interleave, d_trackskew and d_cylskew describe perturbations
        * in the media format used to compensate for a slow controller.
        * Interleave is physical sector interleave, set up by the formatter
        * or controller when formatting.  When interleaving is in use,
        * logically adjacent sectors are not physically contiguous,
        * but instead are separated by some number of sectors.
        *is specified as the ratio of physical sectors traversed
        * per logical sector.  Thus an interleave of 1:1 implies contiguous
        * layout, while 2:1 implies that logical sector 0 is separated
        * by one sector from logical sector 1.
        * d_trackskew is the offset of sector 0 on track N
        * relative to sector 0 on track N-1 on the same cylinder.
        * Finally, d_cylskew is the offset of sector 0 on cylinder N
        * relative to sector 0 on cylinder N-1.
        */
        u_short d_rpm;          /* rotational speed */
        u_short d_interleave;   /* hardware sector interleave */
        u_short d_trackskew;    /* sector 0 skew, per track */
        u_short d_cylskew;      /* sector 0 skew, per cylinder */
        u_long  d_headswitch;   /* head switch time, usec */
        u_long  d_trkseek;      /* track-to-track seek, usec */
        u_long  d_flags;        /* generic flags */
#define NDDATA 5
        u_long  d_drivedata[NDDATA];    /* drive-type specific information */
#define NSPARE 5
        u_long  d_spare[NSPARE];        /* reserved for future use */
        u_long  d_magic2;       /* the magic number (again) */
        u_short d_checksum;     /* xor of data incl. partitions */

        /* filesystem and partition information: */
        u_short d_npartitions;  /* number of partitions in following */
        u_long  d_bbsize;       /* size of boot area at sn0, bytes */
        u_long  d_sbsize;       /* max size of fs superblock, bytes */
        struct  partition {     /* the partition table */
                u_long  p_size; /* number of sectors in partition */
                u_long  p_offset;       /* starting sector */
                u_long  p_fsize;        /* filesystem basic fragment size */
                u_char  p_fstype;       /* filesystem type, see below */
                u_char  p_frag; /* filesystem fragments per block */
                union {
                        u_short cpg;    /* UFS: FS cylinders per group */
                        u_short sgs;    /* LFS: FS segment shift */
                } __partition_u1;
#define p_cpg   __partition_u1.cpg
#define p_sgs   __partition_u1.sgs
                u_short p_cpg;  /* filesystem cylinders per group */
        } d_partitions[MAXPARTITIONS];  /* actually may be more */
```

```
};

/* d_type values: */
#define DTYPE_SMD       1       /* SMD, XSMD; VAX hp/up */
#define DTYPE_MSCP      2       /* MSCP */
#define DTYPE_DEC       3       /* other DEC (rk, rl) */
#define DTYPE_SCSI      4       /* SCSI */
#define DTYPE_ESDI      5       /* ESDI interface */
#define DTYPE_ST506     6       /* ST506 etc. */
#define DTYPE_HPIB      7       /* CS/80 on HP-IB */
#define DTYPE_HPFL      8       /* HP Fiber-link */
#define DTYPE_FLOPPY    10      /* floppy */

#ifdef DKTYPENAMES
static char *dktypenames[] = {
         "unknown",
         "SMD",
         "MSCP",
         "old DEC",
         "SCSI",
         "ESDI",
         "ST506",
         "HP-IB",
         "HP-FL",
         "type 9",
         "floppy",
         0
};
#define DKMAXTYPES      (sizeof(dktypenames) / sizeof(dktypenames[0]) - 1)
#endif

/*
* Filesystem type and version.
* Used to interpret other filesystem-specific
* per-partition information.
*/
#define FS_UNUSED       0       /* unused */
#define FS_SWAP         1       /* swap */
#define FS_V6           2       /* Sixth Edition */
#define FS_V7           3       /* Seventh Edition */
#define FS_SYSV         4       /* System V */
#define FS_V71K         5       /* V7 with 1K blocks (4.1, 2.9) */
#define FS_V8           6       /* Eighth Edition, 4K blocks */
#define FS_BSDFFS       7       /* 4.2BSD fast file system */
#define FS_MSDOS        8       /* MSDOS file system */
#define FS_BSDLFS       9       /* 4.4BSD log-structured file system */
#define FS_OTHER        10      /* in use, but unknown/unsupported */
#define FS_HPFS         11      /* OS/2 high-performance file system */
#define FS_ISO9660      12      /* ISO 9660, normally CD-ROM */
#define FS_BOOT         13      /* partition contains bootstrap */

#ifdef  DKTYPENAMES
static char *fstypenames[] = {
         "unused",
         "swap",
```

```
        "Version 6",
        "Version 7",
        "System V",
        "4.1BSD",
        "Eighth Edition",
        "4.2BSD",
        "MSDOS",
        "4.4LFS",
        "unknown",
        "HPFS",
        "ISO9660",
        "boot",
        0
};
#define FSMAXTYPES      (sizeof(fstypenames) / sizeof(fstypenames[0]) - 1)
#endif

/*
* flags shared by various drives:
*/
#define D_REMOVABLE     0x01    /* removable media */
#define D_ECC           0x02    /* supports ECC */
#define D_BADSECT       0x04    /* supports bad sector forw. */
#define D_RAMDISK       0x08    /* disk emulator */
#define D_CHAIN         0x10    /* can do back-back transfers */

/*
* Drive data for SMD.
*/

#define d_smdflags      d_drivedata[0]
#define D_SSE           0x1     /* supports skip sectoring */
#define d_mindist       d_drivedata[1]
#define d_maxdist       d_drivedata[2]
#define d_sdist         d_drivedata[3]

/*
* Drive data for ST506.
*/
#define d_precompcyl    d_drivedata[0]
#define d_gap3          d_drivedata[1]  /* used only when formatting */

/*
 * Drive data for SCSI.
 */
#define d_blind         d_drivedata[0]

#ifndef LOCORE
/*
* Structure used to perform a format
* or other raw operation, returning data
* and/or register values.
* Register identification and format
* are device- and driver-dependent.
*/
```

```
struct format_op {
        char    *df_buf;
        int     df_count;       /* value-result */
        daddr_t df_startblk;
        int     df_reg[8];      /* result */
};

/*
* Structure used internally to retrieve
* information about a partition on a disk.
*/
struct partinfo {
        struct  disklabel *disklab;
        struct  partition *part;
};

/*
* Disk-specific ioctls.
*/
        /* get and set disklabel; DIOCGPART used internally */
#define DIOCGDINFO   _IOR('d', 101, struct disklabel) /* get */
#define DIOCSDINFO   _IOW('d', 102, struct disklabel) /* set */
#define DIOCWDINFO   _IOW('d', 103, struct disklabel) /* set, update disk */
#define DIOCGPART    _IOW('d', 104, struct partinfo)  /* get partition */

/* do format operation, read or write */
#define DIOCRFORMAT  _IOWR('d', 105, struct format_op)
#define DIOCWFORMAT  _IOWR('d', 106, struct format_op)

#define DIOCSSTEP    _IOW('d', 107, int) /* set step rate */
#define DIOCSRETRIES _IOW('d', 108, int) /* set # of retries */
#define DIOCWLABEL   _IOW('d', 109, int) /* write en/disable label */
#define DIOCSBAD     _IOW('d', 110, struct dkbad) /* set kernel dkbad */
#endif LOCORE
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`disklabel`(1M), `disktab`(4CC)

NAME

disktab – disk description file

SYNOPSIS

```
#include <disktab.h>
```

DESCRIPTION

`/etc/disktab` is a simple database which describes disk geometries and disk partition characteristics. It is used by `disklabel`(1M) to initialize the disk label on the disk. Entries in `disktab` consist of a number of fields separated by a colon, `:`. The first entry for each disk gives the names which are known for the disk, separated by pipe `|` characters. The last name given should be a long name fully identifying the disk.

The following list indicates the normal values stored for each disk entry.

```
Sy Name     Type    Description
(ty       str     Type of disk (e.g. removable, winchester))
(dt       str     Type of controller (e.g.)SMD , ESDI , floppy)
(ns       num     Number of sectors per track)
(nt       num     Number of tracks per cylinder)
(nc       num     Total number of cylinders on the disk)
(sc       num     Number of sectors per cylinder, nc*nt default)
(su       num     Number of sectors per unit, sc*nc default)
(se       num     Sector size in bytes,)
DEV_BSIZE default
(sf       bool    Controller supports bad144-style bad sector forwarding)
(rm       num     Rotation speed, rpm, 3600 default)
(sk       num     Sector skew per track, default 0)
(cs       num     Sector skew per cylinder, default 0)
(hs       num     Headswitch time, usec, default 0)
(ts       num     One-cylinder seek time, usec, default 0)
(il       num     Sector interleave (n:1), 1 default)
(d[0-4]   num     Drive-type-dependent parameters)
(bs       num     Boot block size, default)
BBSIZE
(sb       num     Superblock size, default)
SBSIZE
(ba       num     Block size for partition ‘a’ (bytes))
(bd       num     Block size for partition ‘d’ (bytes))
(be       num     Block size for partition ‘e’ (bytes))
(bf       num     Block size for partition ‘f’ (bytes))
(bg       num     Block size for partition ‘g’ (bytes))
(bh       num     Block size for partition ‘h’ (bytes))
(fa       num     Fragment size for partition ‘a’ (bytes))
(fd       num     Fragment size for partition ‘d’ (bytes))
(fe       num     Fragment size or partition ‘e’ (bytes))
(ff       num     Fragment size for partition ‘f’ (bytes))
(fg       num     Fragment size for partition ‘g’ (bytes))
(fh       num     Fragment size for partition ‘h’ (bytes))
(oa       num     Offset of partition ‘a’ in sectors)
(ob       num     Offset of partition ‘b’ in sectors)
(oc       num     Offset of partition ‘c’ in sectors)
(od       num     Offset of partition ‘d’ in sectors)
(oe       num     Offset of partition ‘e’ in sectors)
(of       num     Offset of partition ‘f’ in sectors)
(og       num     Offset of partition ‘g’ in sectors)
```

```
(oh       num     Offset of partition ‘h’ in sectors)
(pa       num     Size of partition ‘a’ in sectors)
(pb       num     Size of partition ‘b’ in sectors)
(pc       num     Size of partition ‘c’ in sectors)
(pd       num     Size of partition ‘d’ in sectors)
(pe       num     Size of partition ‘e’ in sectors)
(pf       num     Size of partition ‘f’ in sectors)
(pg       num     Size of partition ‘g’ in sectors)
(ph       num     Size of partition ‘h’ in sectors)
(ta       str     Partition type of partition ‘a’)
(tb       str     Partition type of partition ‘b’)
(tc       str     Partition type of partition ‘c’)
(td       str     Partition type of partition ‘d’)
(te       str     Partition type of partition ‘e’)
(tf       str     Partition type of partition ‘f’)
(tg       str     Partition type of partition ‘g’)
(th       str     Partition type of partition ‘h’)
```

FILES

`/etc/disktab`

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`getdiskbyname`(3POSIX), `disklabel`(4CC), `disklabel`(1M), `newfs`(1M)

NAME

exports – define remote mount points for NFS mount requests

SYNOPSIS

`/etc/exports`

FEATURES

NFS_SERVER

DESCRIPTION

The *exports* file specifies remote mount points for the *NFS* mount protocol as defined in the *NFS* server specification; see "Network File System Protocol Specification RFC 1094, Appendix A" .

Each line in the file (other than comment lines that begin with a #) specifies the mount point(s) and export flags within one local server filesystem for one or more hosts. A host may be specified only once for each local filesystem on the server and there may be only one default entry for each server filesystem that applies to all other hosts. The latter exports the filesystem to the “world” and should be used only when the filesystem contains public information.

In a mount entry, the first field(s) specifies the directory path(s) within a server filesystem that can be mounted onto by the corresponding client(s). There are two forms of this specification. The first is to list all mount points as absolute directory paths separated by whitespace. The second is to specify the pathname of the root of the filesystem followed by the *-alldirs* flag; this form allows the host(s) to mount any directory within the filesystem. The pathnames must not have any symbolic links in them and should not have any "." or ".." components. Mount points for a filesystem may appear on multiple lines, each with different sets of hosts and export options.

The second component of a line specifies how the filesystem is to be exported to the host set. The option flags specify whether the filesystem is exported read-only or read-write, and how the client uid is mapped to user credentials on the server.

Export options are specified as follows:

-maproot=<uid>

The credential of the specified user is used for remote access by root. The credential includes all the groups of which the user is a member on the local machine. The user may be specified by name or number. On top of ChorusOS numbers only are supported, Neither */etc/passwd* or */etc/groups* are supported.

-maproot=<uid>:<gid0>:<gid1>:<gid2>

The colon (“:”) separated list is used to specify the precise credentials to be used for remote access by root. The elements of the list may be either names or numbers. On top of ChorusOS names are not supported, only numbers. Note that “user:” should be used to distinguish a credential containing no groups from a complete credential for that user.

-mapall=<uid> or *-mapall=<uid>:<gid0>:<gid1>:<gid2>*

specifies a mapping for all client uids (including root) using the same semantics as *maproot*.

The option *-r* is a synonym for *maproot* in order to be backward compatible with older export file formats.

In the absence of the *maproot* and *mapall* options, remote accesses by root will result in using a credential of -2:-2. All other users will be mapped to their remote credential. If a *maproot* option is given, remote access by root will be mapped to that credential instead of -2:-2. If a *mapall* option is given, all users (including root) will be mapped to that credential instead of their own.

The *kerb* option specifies that the Kerberos authentication server should be used to authenticate and map client credentials.

The *-ro* option specifies that the filesystem should be exported read-only (default read/write). The option *-o* is a synonym for *-ro* in order to be backward compatible with older export file formats.

The third component of a line specifies the host set to which the line applies. The set may be specified in three ways. The first way is to list the host name(s) separated by white space. (Standard internet "dot" addresses may be used instead of names.) The second way is to specify a "netgroup" as defined in the netgroup file (see `netgroup`). The third way is to specify an internet subnetwork using a network and network mask that is defined as the set of all hosts with addresses within the subnetwork. This latter approach requires less overhead within the kernel, and is recommended for cases where the export line refers to a large number of clients within an administrative subnet.

The first two cases are specified by simply listing the name(s) separated by whitespace. All names are checked to see if they are "netgroup" names first, and are assumed to be hostnames otherwise. Using the full domain specification for a hostname can usually circumvent the problem of a host that has the same name as a netgroup. The third case is specified by the flag *-network*=<*netname*> and, optionally, *-mask*=<*netmask*>. If the mask is not specified, it will default to the mask for that network class (A, B or C; see *inet(7P)*).

This third component specifying host(s) and netgroup is not yet supported on top of ChorusOS.

For example:

```
/usr /usr/local -maproot=0:10 friends
/usr -maproot=daemon grumpy.cis.uoguelph.ca 131.104.48.16
/usr -ro -mapall=nobody
/u -maproot=bin: -network 131.104.48 -mask 255.255.255.0
/u2 -maproot=root friends
/u2 -alldirs -kerb -network cis-net -mask
/mnt0 -alldirs -maproot=0:1
```

Given that */usr* , */u* and */u2* are local filesystem mount points, the above example specifies the following: */usr* is exported to host *friends*, where friends is specified in the netgroup file with users mapped to their remote credentials, and root is mapped to uid 0 and group 10. It is exported read-write and the hosts in "friends" can mount either /usr or /usr/local. It is exported to *131.104.48.16* and *grumpy.cis.uoguelph.ca* with users mapped to their remote credentials, and root mapped to the user and groups associated with "daemon". It is exported to the rest of the world as read-only, with all users mapped to the user and groups associated with "nobody".

The */u* mount point is exported to all hosts on the subnetwork *131.104.48* with the root mapped to the uid for "bin", and with no group access.

The */u2* mount point is exported to the host in "friends" with the root mapped to uid and groups associated with "root". It is exported to all hosts on the network "cis-net", allowing mounts at any directory within /u2, and mapping all uids to credentials for the principal that is authenticated by a Kerberos ticket.

The */mnt0* mount point is exported for everyone and root credentials are mapped to uid=0 and gid=1. All other users are mapped to their own credentials. This configuration is typically used with ChorusOS.

FILES

`/etc/exports` The default remote mount-point file.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`mountd`(1M), `nfsd`(1M), `showmount`(1M), `netgroup`(4CC)

BUGS

The export options are tied to the local mount points in the kernel and must be non-contradictory for any exported subdirectory of the local server mount point. It is recommended that all exported directories within the same server filesystem be specified on adjacent lines going down the tree. You cannot specify a hostname that is also the name of a netgroup. Specifying the full domain specification for a hostname can usually circumvent this problem.

NAME

fstab – static information about the filesystems

SYNOPSIS

#include <fstab.h>

DESCRIPTION

The file `fstab` contains descriptive information about the various file systems. It is only read by programs, and not written to; it is the duty of the system administrator to properly create and maintain this file. Each filesystem is described on a separate line; fields on each line are separated by tabs or spaces. The order of records in `fstab` is important because `fsck`(1M) iterates sequentially iterate through `fstab`.

The first field, *fs_spec*, describes the block special device or remote filesystem to be mounted. For filesystems of type UFS, the special file name is the block special file name, and not the character special file name. If a program needs the character special file name, the program must create it by appending an “r” after the last “/” in the special file name.

The second field, *fs_file*, describes the mount point for the filesystem. For swap partitions, this field should be specified as “none”.

The third field, *fs_vfstype*, describes the type of filesystem. The system currently supports the following types of filesystems:

ufs	a local UNIX filesystem
msdosfs	a local DOS filesystem, see RESTRICTIONS for ChorusOS.
nfs	a Sun Microsystems compatible Network File System
swap	a disk partition to be used for swapping

The fourth field, *fs_mntops*, describes the mount options associated with the filesystem. It is formatted as a list of options separated by commas (“,”). It contains at a minimum, the type of mount (see *fs_type* below) plus any additional options appropriate to the filesystem type.

If the options “userquota” or “groupquota” are specified, the filesystem is automatically processed by the `quotacheck` (see RESTRICTIONS for ChorusOS) command, and user and/or group disk quotas are enabled using `quotaon`. By default, filesystem quotas are maintained in files named `quota.user` and `quota.group` which are located at the root of the associated filesystem. These defaults may be overridden by putting an equal sign and an alternative absolute pathname following the quota option. Thus, if the user quota file for `/tmp` is stored in `/var/quotas/tmp.user`, this location can be specified as: `userquota=/var/quotas/tmp.user`.

The type of mount is extracted from the *fs_mntops* field and stored separately in the *fs_type* field (it is not deleted from the *fs_mntops* field). If *fs_type* is “rw” or “ro”, the filesystem whose name is given in the *fs_file* field should be mounted read-write or read-only on the special file specified. If *fs_type* is “sw”, the special

file is made available as a piece of swap space by the swapon(1M) command at the end of the system reboot procedure. Fields other than *fs_spec* and *fs_type* are not used (see RESTRICTIONS for ChorusOS). If *fs_type* is specified as "xx" the entry is ignored. This is useful for showing disk partitions which are currently unused.

The fifth field, fs_freq, is used for filesystems by the dump (see RESTRICTIONS for ChorusOS) command to determine which filesystems need to be dumped. If the fifth field is not present, a value of zero is returned and dump will not dump the filesystem.

The sixth field, *fs_passno*, is used by the fsck(1M) program to determine the order in which filesystem checks are done at reboot time. The root filesystem should be specified with an *fs_passno* of 1, and other filesystems should have an *fs_passno* of 2. Filesystems within a drive will be checked sequentially, but filesystems on different drives will be checked at the same time to use the parallelism available in the hardware. If the sixth field is not present or zero, a value of zero is returned and fsck will not check the filesystem.

```
#define FSTAB_RW   "rw"   /* read-write device */
#define FSTAB_RO   "ro"   /* read-only device */
#define FSTAB_SW   "sw"   /* swap device */
#define FSTAB_XX   "xx"   /* ignore totally */
struct fstab {
    char *fs_spec;    /* block special device name */
    char *fs_file;    /* filesystem path prefix */
    char *fs_vfstype; /* type of filesystem */
    char *fs_mntops;  /* comma separated mount options */
    char *fs_type;    /* rw, ro, sw, or xx */
    int  fs_freq;     /* dump frequency, in days */
    int  fs_passno;   /* pass number on parallel dump */
};
```

FILES

/etc/fstab

HISTORY

This file format appeared in 4.0 BSD.

RESTRICTIONS FOR ChorusOS

Utilities for checking quotas, quotacheck and quotaon, are not implemented.

swapon(1M) is implemented as a built-in C_INIT(1M) system actor command.

dump is not implemented.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

NAME

hosts – host name data base

SYNOPSIS

`/etc/hosts`

DESCRIPTION

The hosts file contains information regarding the known hosts on the TCP/IP. For each host a single line should be present with the following information:

Internet-address official-host-name aliases

Items are separated by any number of blanks and/or TAB characters. A '#' at the beginning of a line indicates a comment; comments are not interpreted by routines which search the file.

Network addresses are specified using the conventional dot ('.') notation using the `inet_addr`(3STDC) routine from the Internet address manipulation library, `inet`(3STDC). Host names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

EXAMPLES

Here is a typical line from the /etc/hosts file:

```
192.33.15.149        raga           # ChorusOS target
```

FILES

/etc/hosts

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`inet`(3STDC)

NAME

netgroup – defines network groups

SYNOPSIS

`/etc/netgroup`

DESCRIPTION

The `netgroup` file specifies “netgroups”, which are sets of (*host*, *user*, *domain*) combinations that are to be given similar network access.

Each line in the file consists of a netgroup name followed by a list of the members of the netgroup. Each member can be either the name of another netgroup or a specification of a combination as follows: (*host*, *user*, *domain*) where the *host*, *user*, and *domain* are character string names for the corresponding component. Any of the comma-separated fields may be empty to specify a “wildcard” value, or may consist of the string “-” to specify “no valid value”. The members of the list may be separated by whitespace or commas; the “\” character may be used at the end of a line to specify line continuation. The functions specified in `getnetgrent`(3POSIX) should normally be used to access the `netgroup` database.

Lines that begin with “#” are treated as comments.

NIS/YP INTERACTION

On most platforms, netgroups are only used in conjunction with NIS and local `/etc/netgroup` files are ignored. With ChorusOS, netgroups can be used with either NIS or local files, but there are certain caveats to consider. The existing netgroup system is extremely inefficient where `innetgr`(3POSIX) lookups are concerned since netgroup memberships are computed on the fly. By contrast, the NIS netgroup database consists of three separate maps (netgroup, netgroup.byuser and netgroup.byhost) that are keyed to allow `innetgr`(3POSIX) lookups to be done quickly. The ChorusOS netgroup system can interact with the NIS netgroup maps in the following ways:

- If the `/etc/netgroup` file does not exist, or it exists and is empty, or it exists and contains only a “+” , and NIS is running, netgroup lookups will be done exclusively through NIS, with `innetgr`(3POSIX) taking advantage of the netgroup.byuser and netgroup.byhost maps to speed up searches. (This is more or less compatible with the behavior of SunOS and similar platforms.)
- If the `/etc/netgroup` exists and contains only local netgroup information (with no NIS “+” token), then only the local netgroup information will be processed (and NIS will be ignored).
- If `/etc/netgroup` exists and contains both local netgroup data and the NIS “+” token, the local data and the NIS netgroup map will be processed as a single combined netgroup database. While this configuration is the most flexible, it is also the least efficient: in particular, `innetgr`(3POSIX) lookups will be especially slow if the database is large.

FILES

`/etc/netgroup` the `netgroup` database

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`getnetgrent`(3POSIX), `exports`(4CC)

COMPATIBILITY

This file format is compatible with that of a number of vendors; note, however, that not all vendors use an identical format.

BUGS

The interpretation of access restrictions based on the members of a netgroup is left up to the network applications. The behavior of the domain specification with regard to the BSD environment is undefined.

NAME

networks – network name data base

DESCRIPTION

The *networks* file contains information regarding the known networks which comprise the DARPA Internet. For each network, a single line should be present containing the following information:

<official network name> <network number> <aliases>

Items are separated by any number of blanks and/or tab characters. A "#" at the beginning of a line indicates a comment; characters up to the end of that line are not interpreted by routines which search the file. This file is usually created from the official network data base maintained at the Network Information Control Center (NIC) , local changes may be required to bring it up to date for unofficial aliases and/or unknown networks.

Network numbers may be specified using the conventional dot (".") notation using the *inet_network(3STDC)* routine from the Internet address manipulation library, *inet(3STDC)*. Network names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

`/etc/networks`

The *networks* file resides in */etc*.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`getnetent`(3POSIX)

BUGS

A name server should be used instead of a static file.

NAME

protocols – protocol name data base

DESCRIPTION

The *protocols* file contains information regarding the known protocols used in the DARPA Internet. For each protocol, a single line should be present containing the following information:

<*official protocol name*> <*protocol number*> <*aliases*>

Items are separated by any number of blanks and / or tab characters. A "#" at the beginning of a line indicates a comment; characters up to the end of that line are not interpreted by routines which search the file.

Protocol names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

`/etc/protocols`

The *protocols* file resides in */etc*.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`getprotoent`(3POSIX)

BUGS

A name server should be used instead of a static file.

NAME

rc.chorus – ChorusOS system initialization file

SYNOPSIS

`/etc/rc.chorus.`*target.system.IP.address*

`/etc/rc.chorus`

DESCRIPTION

`rc.chorus` is used by the `C_INIT`(1M) supervisor actor to customize system initialization as soon as the ChorusOS root file system containing `/etc/rc.chorus.`*target.system.IP.address* is mounted, often by one of the last commands in `sysadm.ini`(4CC). The *target.system.IP.address* suffix enables multiple target systems requiring customized system intialization procedures to share a common root file system exported through NFS by a single server.

Each line in `rc.chorus` is a command for the command interpreter, such as:

```
arun /bin/inetNShost.O &
```

Many system initialization commands are included in `sysadm.ini`. `rc.chorus` provides the mechanism to continue and extend system initialization after the root file system is mounted.

FILES

`/etc/rc.chorus.`*target.system.IP.address*

`/etc/rc.chorus`

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`C_INIT`(1M), `sysadm.ini`(4CC)

NOTES

If `/etc/rc.chorus.`*target.system.IP.address* is not found, the `C_INIT` system actor attempts to execute `/etc/rc.chorus`.

NAME

resolv.conf – configuration file for the domain name system resolver

SYNOPSIS

`/etc/resolv.conf`

DESCRIPTION

The resolver configuration file contains information that is read by the *inetNSdns* Internet DOMAIN Name Server. This file need only be created to specify an explicit default domain name or to specify name servers to use on other machines. The file is designed to be readable by users and contains a list of keyword-value pairs that provide various types of resolver information.

keyword value

The different configuration options are:

`nameserver address`
: The Internet address (in dot notation) of a name server that the resolver should query. Up to MAXNS (currently 3) name servers may be listed, the resolver library tries them in the order listed. The policy used is to try a name server, if the query times out, try the next, until the end of the list is reached; then repeat trying all the name servers until a maximum number of retries have been performed.

`domain name`
: The default domain to append to names that do not have a dot in them, which are to be used in searches.

search
: Search list for hostname lookup. The search list is normally determined from the local domain name; by default, it begins with the local domain name, then successive parent domains that have at least two components in their names. This may be changed by listing the desired domain search path following the search keyword with spaces or tabs separating the names. Most resolver queries will be attempted using each component of the search path in turn until a match is found. Note that this process may be slow and will generate a lot of network traffic if the servers for the listed domains are not local, and that queries will time out if no server is available for one of the domains.

  The `domain` and `search` keywords are mutually exclusive. If more than one instance of these

keywords is present, the last instance will override.

The search list is currently limited to six domains with a total of 256 characters.

The *keyword-value* pair must appear on a single line, and the *keyword* (for instance, `nameserver`) must start the line. The *value* follows the *keyword*, separated by white space.

FILES

/etc/resolv.conf

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`hostname`(1CC), `inetNS`(1M), `gethostbyname`(3STDC)

NAME

security – user security file

SYNOPSIS

`/etc/security`

DESCRIPTION

The security file is used by `C_INIT`(1M) and has two main purposes:
- authentication of the remote user and remote host which issued the `rsh` command to communicate with `C_INIT`
- initialization of credentials and privilege that the user will have during the execution of the `C_INIT`(1M) built-in command. In general, the credentials will be the same as the remote user has on the remote host.

Each line of the file has the following format:

username:trusted:uid:gid:add_groups_list:rem_hosts_list:password_not_encrypted

where:

username — The user's name.

This field corresponds to the remote user's login name on the remote host. The wild character '*' may be used to describe all other users.

trusted — The user's privilege.

If this field is the "TRUSTED" string, the user is trusted and has access to the full set of `C_INIT` commands. In any other case, the user is declared as non-trusted, and has restricted access to this set of commands.

uid — The user's numerical ID.

If this field is empty, the user's id will take the default configuration value; see `C_INIT`(1CC).

gid — The group's numerical ID.

If this field is empty, the group's id will take the default configuration value; see `C_INIT`(1CC).

add_groups_list — Additional groups list.

This field is a list of supplementary groups, separated by commas. It may be empty.

rem_hosts_list — Authorized remote hosts list.

This field contains the list of remote hosts authorized to issue `C_INIT` commands for the user *username*. Items are separated by commas,

and hosts are identified by addresses using the dot (“.”) Internet notation. The wild character “*” denotes all hosts. You can explicitly deny the access to a user by leaving this field empty.

password_not_encrypted Password for *username*.

This field contains a single string representing the password for *username*. Note that this field is not encrypted, so if a user has access to the `/etc/security` file, they can read all passwords.

If the field is empty, the password is `NULL`. Some utilities, such as `ftpd`(1M) cannot enable secure mode for users who do not have passwords.

A hash (“#”) at the beginning of a line indicates a comment; comments are not interpreted.

EXAMPLES

A security file could look something like this:

```
smith:TRUSTED:256:300::*          # trusted user from any host
john::257:300:301,302:192.33.12.1 # non trusted from a particular host
foobar::258:305:                  # non authorized user
*:::::*                           # all others: non trusted from any host
```

WARNINGS

The authentication procedure succeeds or fails as soon as a matching user entry is found. The order of entries is therefore important.

FILES

/etc/security

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`C_INIT`(1M)

BUGS

A trusted user who is not root has root permissions when using the built-in `C_INIT`(1M) command, `ifconfig`. This is not the case when using the `ifconfig.r` actor.

NAME

services – service name data base

DESCRIPTION

The *services* file contains information regarding the known services available on the *DARPA* Internet. For each service, a single line should be present containing the following information: *service_name port_number/protocol_name aliases*

Items are separated by any number of blanks and / or tab characters. The port number and protocol name are considered to be a single item ; a " / " is used to separate the port and protocol (for example, "512 / tcp"). A "#" at the beginning of a line indicates a commnet; subsequent characters up to the end of the line are not interpreted by the routines which search the file. Service names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/services The *services* file resides in */etc* .

BUGS

A name server should be used instead of a static file.

HISTORY

This file format appeared in 4.2BSD .

RESTRICTIONS FOR ChorusOS

Currently, this file is only used with `ftp` and `telnet` clients. The only relevant information concerns getting the port number of the ftp protocol: *ftp 21/tcp,* and of telnet protocol: *ftp 23/tcp* .

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`getservent`(3POSIX)

NAME

sysadm.ini – initializes file and network devices at boot time

SYNOPSIS

`/image/sys_bank/sysadm.ini`

DESCRIPTION

`sysadm.ini` is scanned by `C_INIT` in order to execute the commands in `sysadm.ini` at boot time. Thus, `sysadm.ini` is generally used to initialize file and network devices. `sysadm.ini` is itself embedded in the system image and is read from the `/image/sys_bank` directory.

All `C_INIT` commands such as `mkdev`, `mknod`, `ifconfig`, `route` and `mount` may be included in `sysadm.ini`. Commands may also run any executables included in the system image at build time, using the syntax:

arun *executable* [*arguments*]

`sysadm.ini` may therefore be adapted to perform all initialization required by applications.

NETWORK INITIALIZATION

The following list shows the network interfaces that can be set up using the `mkdev` command. Each interface configured corresponds to a structure maintained by the `IOM` component.

ifeth*N*	Ethernet interface
lo*N*	loopback interface
tty*N*	tty interface
sl*N*	SLIP interface
ppp*N*	PPP interface
bpf*N*	BSD packet filter interface.

Where *N* is a digit, such as `0`, `1`, `2`, and so forth.

Note that only `ifeth` and `tty` interfaces connect directly to hardware devices (in this case, Ethernet and serial lines, respectively). `ppp` and `sl` interfaces rely on a `tty` interface for the serial line, and `bpf` and `lo` interfaces are logical.

DEVICE INITIALIZATION

The following list shows the file and network devices you can set up using the `mknod` command. Each interface you configure corresponds to a node embedded in the system image and visible in the `/dev` directory after system initialization.

/dev/hd0*X*	IDE disk, block access
/dev/rhd0*X*	IDE disk, raw access
/dev/rd0*X*	RAM disk, block access
/dev/rrd0*X*	RAM disk, raw access
/dev/flash	flash disk, block access
/dev/rflash	flash disk, raw access

/dev/sd0*X*	SCSI disk, block access
/dev/rsd0*X*	SCSI disk, raw access
/dev/tty0*M*	tty, raw access
/dev/ttyp*M*	pseudo-tty, slave
/dev/ptyp*M*	pseudo-tty, master
/dev/bpf*N*	BSD packet filter device

Where *N* is a digit, such as `0`, `1`, `2`, ..., and *M* is a digit, such as `1`, `2`, `3`, ..., and *X* is a single letter referring to a partition, such as `a`, `b`, ..., `h`.

An example `sysadm.ini` file is provided below.

EXAMPLES

Setting Up an Ethernet Interface

```
mkdev  ifeth 0 /pci/epic100
```

The first argument is the interface name. The second argument identifies the unit number. The real interface name is obtained by concatenating the two — `ifeth0` in this case.

The third argument identifies the hardware device you want to connect to the interface. The third argument is not mandatory. If you type:

```
mkdev  ifeth 0
```

`mkdev` takes the first hardware device available in the device tree.

Setting Up a Loopback Interface

```
mkdev  lo 0
```

The loopback interface is logical, so it does not depend on hardware. Therefore, the `mkdev` command does not require a device tree pathname. The result is a `lo0` interface.

Setting Up a Serial Interface for PPP

Note that this example

First, set up a tty device managing the second target serial port (the first one is used for the `tip` line):

```
mkdev tty  0
#mkdev tty  0 /pci/pci-isa/ns16550-2 # x86, Pentium
#mkdev tty  0 /raven/w83c553/ns16550-2 # PowerPC
```

Next, set up a PPP network interface:

```
mkdev ppp  0
```

The `ppp0` interface is bound later to a tty device by the PPP manager. Therefore `mkdev` does not require a device tree pathname.

Finally, set up one or several pseudo devices so that the PPP daemon can access the tty and configure it for PPP. Use `mknod` for this. Start by creating a node to access the tty:

```
mknod  /dev/tty01 c 0 0
```

which is sufficient for normal PPP daemon mode. However, if you use the dial-up on demand options of PPP, you need the so-called *pseudo-ttys* configured as follows:

```
mknod /dev/ttyp0 c      5 0
mknod /dev/ptyp0 c      6 0
```

which are used by PPP to detect when IP traffic occurs, before it has actually configured the tty interface.

Configuring the IP Network

Once all the devices and interfaces are set up, you can execute various network administration commands to initialize your network. The following list provides brief descriptions of the most important methods. See the man pages for details.

ifconfig *ifname IP_ADDRESS* netmask *MASK* broadcast *BROADCAST*
: The simplest method. Caution is recommended however, as picking a wrong IP address can cause network problems. `ifconfig` directly sets up the interface, host address, netmask and broadcast address.

rarp *ifname*
: `rarp` obtains the IP address of the target from a RARP server, and then executes `ifconfig`. Therefore, you do not need to add an `ifconfig` in `sysadm.ini`. In order to use RARP, set the `ADMIN_RARP` and `BPF` features to `true`. You also need to set up a `bpf` device:

```
mkdev bpf 0
mknod /dev/bpf0 c 23 0
```

arun dhclient *ifname* &
: `dhclient` enables you to obtain your IP address as well as other parameters such as routing and masks using the DHCP protocol. To use `dhclient`, you must set the `BPF` feature to `true`, include `dhclient.r` in the system image, and include the following in `sysadm.ini`:

```
mkdev bpf 0
mknod /dev/bpf0 c 23 0
arun /image/sys_bank/dhclient ifeth0 &
ifwait ifeth0
```

You must run `dhclient` in the background, using `&`, to prevent `C_INIT` from being blocked. `ifwait` allows you to wait for the interface to be `UP`. This is important because `dhclient` may run for a long time before obtaining an IP address. Unless you use `ifwait`, subsequent commands may execute on an unconfigured network.

As does RARP, `dhclient` executes the `ifconfig` command itself. Do not forget to ask your system administrator to add an entry for the target to the DHCP server database so that the target receives replies to DHCP requests.

Mounting a Root File System

Once all the devices and network interfaces are set up, you can mount a root file system to access actors that would not fit or that you did not want to include in the system image. The following example mounts an NFS file system located on the system with IP address, `123.45.67.89`:

```
# dhclient built into the system image
arun /image/sys_bank/dhclient ifeth0 &
ifwait ifeth0
mount 123.45.67.89:/export/ChorusOS/root /
arun /bin/myactor # my actor is on the host
```

Example
`sysadm.ini`

```
# set umask to 0 during system configuration
umask 0

#
# First build the interfaces you need using the mkdev(1M) command.
# All interfaces require a unit number. For bpf and tty, this number
# also corresponds to the device minor. Thus, a
#     mkdev tty  0
# corresponds to a
#     mknod  /dev/tty01 major 0
#
# Some interfaces, such as Ethernet and tty, also require an
# associated hardware device.
#

# Ethernet
mkdev  ifeth 0

# loopback interface
mkdev  lo 0

# If you need a tty for a serial line, note that the first device
# Is reserved for a tip(1) line. As the mkdev(1M) command uses the
# first free device it finds, it is not necessary to specify the
# device, however.
#
#mkdev tty 0
#mkdev tty 0 /pci/pci-isa/ns16550-2   # x86, Pentium
#mkdev tty 0 /raven/w83c553/ns16550-2 # PowerPC

# If you want SLIP and/or PPP ifnets,
#mkdev  sl 0
#mkdev  ppp 0

# rarp(1M) needs bpf. The unit number corresponds to the device minor.
mkdev bpf 0
mkdev bpf 1

#
# Next, build all the nodes for accessing devices.
```

```
#

# 0: Raw tty devices
mknod /dev/tty01 c      0 0
#mknod /dev/tty02 c     0 1

# 1: reserved for local console

# 2: pseudo devs.
mknod /dev/mem c        2 0
mknod /dev/kmem c       2 1
mknod /dev/null c       2 2
mknod /dev/zero c       2 3

# 3: ST506/ESDI/IDE disk 0
mknod /dev/rhd0a c      3 0
mknod /dev/rhd0b c      3 1
mknod /dev/rhd0c c      3 2
mknod /dev/rhd0d c      3 3
mknod /dev/rhd0e c      3 4
mknod /dev/rhd0f c      3 5
mknod /dev/rhd0g c      3 6
mknod /dev/rhd0h c      3 7

# 3: ST506/ESDI/IDE disk 1
mknod /dev/rhd1a c      3 8
mknod /dev/rhd1b c      3 9
mknod /dev/rhd1c c      3 10
mknod /dev/rhd1d c      3 11
mknod /dev/rhd1e c      3 12
mknod /dev/rhd1f c      3 13
mknod /dev/rhd1g c      3 14
mknod /dev/rhd1h c      3 15

# 4: ST506/ESDI/IDE disk 0
mknod /dev/hd0a b       4 0
mknod /dev/hd0b b       4 1
mknod /dev/hd0c b       4 2
mknod /dev/hd0d b       4 3
mknod /dev/hd0e b       4 4
mknod /dev/hd0f b       4 5
mknod /dev/hd0g b       4 6
mknod /dev/hd0h b       4 7

# 4: ST506/ESDI/IDE disk 1
mknod /dev/hd1a b       4 8
mknod /dev/hd1b b       4 9
mknod /dev/hd1c b       4 10
mknod /dev/hd1d b       4 11
mknod /dev/hd1e b       4 12
mknod /dev/hd1f b       4 13
mknod /dev/hd1g b       4 14
mknod /dev/hd1h b       4 15

# 5: ptys
```

```
mknod /dev/ttyp0 c      5 0
mknod /dev/ttyp1 c      5 1

# 6: ptys
mknod /dev/ptyp0 c      6 0
mknod /dev/ptyp1 c      6 1

# 7: Flash Memory
mknod /dev/rflash0a c   7 0
mknod /dev/rflash0b c   7 1
mknod /dev/rflash0c c   7 2

# 8: Flash Memory
mknod /dev/flash0a b    8 0
mknod /dev/flash0b b    8 1
mknod /dev/flash0c b    8 2

# 9: SCSI disk
mknod /dev/rsd0a c      9 0
mknod /dev/rsd0b c      9 1
mknod /dev/rsd0c c      9 2
mknod /dev/rsd0d c      9 3
mknod /dev/rsd0e c      9 4
mknod /dev/rsd0f c      9 5
mknod /dev/rsd0g c      9 6
mknod /dev/rsd0h c      9 7

#10 available for SCSI disk block
mknod /dev/sd0a b       10 0
mknod /dev/sd0b b       10 1
mknod /dev/sd0c b       10 2
mknod /dev/sd0d b       10 3
mknod /dev/sd0e b       10 4
mknod /dev/sd0f b       10 5
mknod /dev/sd0g b       10 6
mknod /dev/sd0h b       10 7


# 11-12: reserved for the boot RAM disk for archive

# 13: RAM disk
mknod /dev/rrd0a c      13 0
mknod /dev/rrd0b c      13 1
mknod /dev/rrd0c c      13 2

# 14: RAM disk
mknod /dev/rd0a b       14 0
mknod /dev/rd0b b       14 1
mknod /dev/rd0c b       14 2

#15-16 : available for cdrom

# 23: BPF
mknod /dev/bpf0 c       23 0
mknod /dev/bpf1 c       23 1
```

```
# set umask to default value after system configuration
umask 22

#
# Initialize the network.
#

# Using ifconfig
#ifconfig ifeth0 YOUR.TARGET.IP.ADDRESS  netmask 0xffff0000 broadcast 129.157.255.255

# Using rarp(1M) (requires ADMIN_RARP(5FEA) and BPF(5FEA) features)
rarp ifeth0

# Using DHCP (requires BPF(5FEA) feature)
#
# This also requires that the dhclient(1M) actor be built into the
# system image.
#
#arun /image/sys_bank/dhclient ifeth0 &
## wait for ifeth0 to be UP
#ifwait ifeth0

# Other interfaces
ifconfig lo0 127.0.0.1 up

# Print network interface status to the console.
ifconfig -a

# start the Applicative Debug Server from NFS
#arun /bin/rdbc
# start the Applicative Debug Server from the system image
#arun -xip /image/sys_bank/rdbc

#
# Additional initialization commands
#
#console
rshd
#mount YOUR.HOST.IP.ADDRESS:/PATH-TO-TARGET-ROOT-FILESYSTEM /
```

FILES `/image/sys_bank/sysadm.ini`

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO `C_INIT`(1M), `arun`(1M), `mkdev`(1M), `mknod`(1M), `rc.chorus`(4CC)

NAME

tcl, tclsh, tclsh_u – interpreter for Tcl, an embeddable scripting language

SYNOPSIS

`rsh` *target* `arun tclsh`

`rsh` *target* `arun tclsh_u`

DESCRIPTION

Tcl is a high-level programming language whose syntax resembles that of csh and C.

The supervisor (`tclsh`) and user (`tclsh_u`) actors are Tcl interpreters for ChorusOS.

EXTENDED DESCRIPTION

The Tcl interpreter delivered with ChorusOS allows you to run Tcl scripts directly on target systems in either supervisor or user mode.

In order to use the Tcl interpreter on the target system, you must build the interpreter using the sources provided, and then run the interpreter as an actor on the target system.

You may include a Tcl resource (`.tclshrc`) file for the interpreter in the `HOME` directory. See the example below.

Building the Tcl Interpreter

Do *not* follow the standard instructions for building the Tcl interpreter. Instead, follow the procedure below:

1. Make sure that `ChorusOSMkMf` is in your `PATH` .

```
host% which
ChorusOSMkMfinstallation_directory/target_family-bin/host/bin/ChorusOSMkMf
```

2. Change to the directory where the Tcl sources are located:

```
host% cd ???
```

3. Create the Makefile:

```
host% make Makefile
```

4. Build the interpreter:

```
host% make
```

5. Install the interpreter:

```
host% make install
```

Install copies the files required to use the Tcl interpreter to the locations listed in the table below.

File(s)	Install Location
`tclsh`	`MERGEDIR /bin`
`tclsh_u`	`MERGEDIR /bin`
`libtcl.a`	`MERGEDIR /lib/tcl`
`tcl.h`	`MERGEDIR /include/tcl`
additional files used by `tclsh`	`MERGEDIR /include/tcl`

Running the Tcl Interpreter on the Target System

To download and run a Tcl interpreter on the target system, follow the procedure below:

1. Make sure `/var/tmp` under the exported tree is writable from the target system.
2. If you run `tclsh` as a supervisor actor, make sure your system has enough stack space. `kern.exec.dflSysStackSize = 0x4000` (at least)
3. If you run `tclsh_u` , you may include a resource file (`.tclshrc`) in the `HOME` directory.
4. Run the interpreter:

host% **`rsh`** *target* **`arun tclsh`** *supervisor mode*

or:

host% **`rsh`** *target* **`arun tclsh_u`** *user mode*

EXAMPLES

The example Tcl resource file (`.tclshrc`) below simulates very simple versions of `ls` , `cat` , `cp` and `more` commands. Remember to place the `.tclshrc` file in the `HOME` directory.

```
proc ls args {
   if {$args == ""} {set args "*"}
   foreach i [lsort [glob $args]] {
       puts stdout $i
   }
}

proc cat args {
   set f [open $args]
   set x [read $f]
   puts stdout $x
   close $f
}

proc cp {f1 f2} {
   set f [open $f1]
   set g [open $f2 "w"]
   set x [read $f]
   puts $g $x
   close $f
   close $g
}

proc more args {
   set count 0
   set f [open $args]
   while {[gets $f line] >= 0} {
         incr count 1
         puts stdout $line
         if {$count == 24} {
            puts -nonewline stdout "--More--"
```

```
            gets stdin order
            if {$order == "q"} {
                close $f
                return
            }
            if {$order == " "} {
                set count 0
            } else {
                set count 23
            }
        }
    }
    close $f
}
```

FILES

These Tcl interpreter source files have been adapted for ChorusOS:

- `tcl.h`
- `tclExpr.c`
- `tclGlob.c`
- `tclInt.h`
- `tclMain.c`
- `tclPort.h`
- `tclUnixAZ.c`
- `tclUnixUtil.c`

Modifications to the original FreeBSD sources are under `#ifdef CLASSIX`.

These files have been added:

- `clxStubs.c`
- `Imakefile`
- `tclPort.h`

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

Tcl and the Tk Toolkit, by John Ousterhout, Addison-Wesley, 1994, ISBN 0–201–63337–X.

Practical Programming in Tcl and Tk, by Brent Welch, Prentice-Hall, 1995, ISBN 0–13–182007–9.

NOTES

This release of ChorusOS supports Tcl 7.4.

NAME

tcl, tclsh, tclsh_u – interpreter for Tcl, an embeddable scripting language

SYNOPSIS

`rsh` *target* `arun tclsh`

`rsh` *target* `arun tclsh_u`

DESCRIPTION

Tcl is a high-level programming language whose syntax resembles that of csh and C.

The supervisor (`tclsh`) and user (`tclsh_u`) actors are Tcl interpreters for ChorusOS.

EXTENDED DESCRIPTION

The Tcl interpreter delivered with ChorusOS allows you to run Tcl scripts directly on target systems in either supervisor or user mode.

In order to use the Tcl interpreter on the target system, you must build the interpreter using the sources provided, and then run the interpreter as an actor on the target system.

You may include a Tcl resource (`.tclshrc`) file for the interpreter in the `HOME` directory. See the example below.

Building the Tcl Interpreter

Do *not* follow the standard instructions for building the Tcl interpreter. Instead, follow the procedure below:

1. Make sure that `ChorusOSMkMf` is in your `PATH` .

host% **`which`**
`ChorusOSMkMf`*`installation_directory/target_family`*`-bin/host/bin/ChorusOSMkMf`

2. Change to the directory where the Tcl sources are located:

host% **`cd ???`**

3. Create the Makefile:

host% **`make Makefile`**

4. Build the interpreter:

host% **`make`**

5. Install the interpreter:

host% **`make install`**

Install copies the files required to use the Tcl interpreter to the locations listed in the table below.

File(s)	Install Location
`tclsh`	`MERGEDIR /bin`
`tclsh_u`	`MERGEDIR /bin`
`libtcl.a`	`MERGEDIR /lib/tcl`
`tcl.h`	`MERGEDIR /include/tcl`
additional files used by `tclsh`	`MERGEDIR /include/tcl`

Running the Tcl Interpreter on the Target System

To download and run a Tcl interpreter on the target system, follow the procedure below:

1. Make sure `/var/tmp` under the exported tree is writable from the target system.
2. If you run `tclsh` as a supervisor actor, make sure your system has enough stack space. `kern.exec.dflSysStackSize = 0x4000` (at least)
3. If you run `tclsh_u` , you may include a resource file (`.tclshrc`) in the `HOME` directory.
4. Run the interpreter:

host% **`rsh`** *target* **`arun tclsh`** *supervisor mode*

or:

host% **`rsh`** *target* **`arun tclsh_u`** *user mode*

EXAMPLES

The example Tcl resource file (`.tclshrc`) below simulates very simple versions of `ls` , `cat` , `cp` and `more` commands. Remember to place the `.tclshrc` file in the `HOME` directory.

```
proc ls args {
   if {$args == ""} {set args "*"}
   foreach i [lsort [glob $args]] {
       puts stdout $i
   }
}

proc cat args {
   set f [open $args]
   set x [read $f]
   puts stdout $x
   close $f
}

proc cp {f1 f2} {
   set f [open $f1]
   set g [open $f2 "w"]
   set x [read $f]
   puts $g $x
   close $f
   close $g
}

proc more args {
   set count 0
   set f [open $args]
   while {[gets $f line] >= 0} {
         incr count 1
         puts stdout $line
         if {$count == 24} {
            puts -nonewline stdout "--More--"
```

```
            gets stdin order
            if {$order == "q"} {
                close $f
                return
            }
            if {$order == " "} {
                set count 0
            } else {
                set count 23
            }
        }
    }
    close $f
}
```

FILES

These Tcl interpreter source files have been adapted for ChorusOS:

- `tcl.h`
- `tclExpr.c`
- `tclGlob.c`
- `tclInt.h`
- `tclMain.c`
- `tclPort.h`
- `tclUnixAZ.c`
- `tclUnixUtil.c`

Modifications to the original FreeBSD sources are under `#ifdef CLASSIX`.

These files have been added:

- `clxStubs.c`
- `Imakefile`
- `tclPort.h`

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

Tcl and the Tk Toolkit, by John Ousterhout, Addison-Wesley, 1994, ISBN 0–201–63337–X.

Practical Programming in Tcl and Tk, by Brent Welch, Prentice-Hall, 1995, ISBN 0–13–182007–9.

NOTES

This release of ChorusOS supports Tcl 7.4.

NAME

tcl, tclsh, tclsh_u – interpreter for Tcl, an embeddable scripting language

SYNOPSIS

`rsh` *target* `arun tclsh`

`rsh` *target* `arun tclsh_u`

DESCRIPTION

Tcl is a high-level programming language whose syntax resembles that of csh and C.

The supervisor (`tclsh`) and user (`tclsh_u`) actors are Tcl interpreters for ChorusOS.

EXTENDED DESCRIPTION

The Tcl interpreter delivered with ChorusOS allows you to run Tcl scripts directly on target systems in either supervisor or user mode.

In order to use the Tcl interpreter on the target system, you must build the interpreter using the sources provided, and then run the interpreter as an actor on the target system.

You may include a Tcl resource (`.tclshrc`) file for the interpreter in the `HOME` directory. See the example below.

Building the Tcl Interpreter

Do *not* follow the standard instructions for building the Tcl interpreter. Instead, follow the procedure below:

1. Make sure that `ChorusOSMkMf` is in your `PATH` .

```
host% which
ChorusOSMkMfinstallation_directory/target_family-bin/host/bin/ChorusOSMkMf
```

2. Change to the directory where the Tcl sources are located:

```
host% cd ???
```

3. Create the Makefile:

```
host% make Makefile
```

4. Build the interpreter:

```
host% make
```

5. Install the interpreter:

```
host% make install
```

Install copies the files required to use the Tcl interpreter to the locations listed in the table below.

File(s)	Install Location
`tclsh`	`MERGEDIR /bin`
`tclsh_u`	`MERGEDIR /bin`
`libtcl.a`	`MERGEDIR /lib/tcl`
`tcl.h`	`MERGEDIR /include/tcl`
additional files used by `tclsh`	`MERGEDIR /include/tcl`

Running the Tcl Interpreter on the Target System

To download and run a Tcl interpreter on the target system, follow the procedure below:

1. Make sure `/var/tmp` under the exported tree is writable from the target system.
2. If you run `tclsh` as a supervisor actor, make sure your system has enough stack space. `kern.exec.dflSysStackSize = 0x4000` (at least)
3. If you run `tclsh_u` , you may include a resource file (`.tclshrc`) in the `HOME` directory.
4. Run the interpreter:

host% **`rsh`** *target* **`arun tclsh`** *supervisor mode*

or:

host% **`rsh`** *target* **`arun tclsh_u`** *user mode*

EXAMPLES

The example Tcl resource file (`.tclshrc`) below simulates very simple versions of `ls` , `cat` , `cp` and `more` commands. Remember to place the `.tclshrc` file in the `HOME` directory.

```
proc ls args {
   if {$args == ""} {set args "*"}
   foreach i [lsort [glob $args]] {
       puts stdout $i
   }
}

proc cat args {
   set f [open $args]
   set x [read $f]
   puts stdout $x
   close $f
}

proc cp {f1 f2} {
   set f [open $f1]
   set g [open $f2 "w"]
   set x [read $f]
   puts $g $x
   close $f
   close $g
}

proc more args {
   set count 0
   set f [open $args]
   while {[gets $f line] >= 0} {
         incr count 1
         puts stdout $line
         if {$count == 24} {
            puts -nonewline stdout "--More--"
```

```
            gets stdin order
            if {$order == "q"} {
                close $f
                return
            }
            if {$order == " "} {
                set count 0
            } else {
                set count 23
            }
        }
    }
    close $f
}
```

FILES

These Tcl interpreter source files have been adapted for ChorusOS:

- `tcl.h`
- `tclExpr.c`
- `tclGlob.c`
- `tclInt.h`
- `tclMain.c`
- `tclPort.h`
- `tclUnixAZ.c`
- `tclUnixUtil.c`

Modifications to the original FreeBSD sources are under `#ifdef CLASSIX`.

These files have been added:

- `clxStubs.c`
- `Imakefile`
- `tclPort.h`

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

Tcl and the Tk Toolkit, by John Ousterhout, Addison-Wesley, 1994, ISBN 0–201–63337–X.

Practical Programming in Tcl and Tk, by Brent Welch, Prentice-Hall, 1995, ISBN 0–13–182007–9.

NOTES

This release of ChorusOS supports Tcl 7.4.

Index