Network Time Protocol Reference Manual

Sun Microsystems Computer Company
A Sun Microsystems, Inc. Business
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

Part No: 805-0079-10
Revision A, March 1997

NAME

Intro – NTP commands

DESCRIPTION

This section describes commands of the Network Time Protocol (ntp or xntp).

ntpdate(1M)	set date and time via ntp
ntpq(1M)	standard ntp query program
ntptrace(1M)	trace ntp hosts back to their master time source
tickadj(1M)	fiddle time-related variables in the kernel
xntpd(1M)	network time protocol daemon
xntpdc(1M)	query/control program for the ntp daemon

NAME

ntpdate – set date and time via ntp

SYNOPSIS

ntpdate [−**bdos**] [−**a** *key#*] [−**e** *authdelay*] [−**k** *keyfile*]
[−**o** *version#*] [−**p** *samples*] [−**t** *timeout*] *server ...*

DESCRIPTION

ntpdate sets the local date and time by polling the Network Time Protocol server(s) on the host(s) given as arguments to determine the correct time. It must be run as root on the local host. A number of samples are obtained from each of the servers specified and the standard NTP clock filter and selection algorithms are applied to select the best of these. Typically, **ntpdate** can be inserted in the **/etc/rc.local** startup script to set the time of day at boot time, or it can be run from time–to–time via **cron**(1M). Note that **ntpdate**'s reliability and precision improve dramatically with greater numbers of servers. While a single server may be used, better performance and reliability on the part of any one server are obtained when at least three or four servers, or more, are used.

Time adjustments are made by **ntpdate** in one of two ways. If it determines your clock is off by more than 0.5 seconds it simply steps the time by calling **settimeofday**(3C). If the error is less than 0.5 seconds, **ntpdate**'s default behavior is to slew the clock's time via a call to **adjtime**(2) with the offset.

The latter technique is less disruptive and more accurate when the offset is small, and works quite well when **ntpdate** is run by **cron**(1M) every hour or two.

The adjustment made in the latter case is actually 50% larger than the measured offset since it tends to keep a badly drifting clock more accurate. This is at some expense to stability, though the tradeoff is usually advantageous.

At boot time, however, it is usually better to always step the time.

ntpdate declines to set the date if an NTP server daemon (*e.g.*, **xntpd**(1M)) is running on the same host. When running **ntpdate** on a regular basis from **cron**(1M) as an alternative to running a daemon, doing so once every hour or two results in precise enough timekeeping to avoid stepping the clock.

OPTIONS

−**b** Always step the time.

−**d** Print the date **ntpdate** would set, without letting it set it, and print information useful for general debugging.

−**o** *version#*
Poll as a *version#* implementation of **ntpdate**. By default, **ntpdate** claims to be NTP version 2 in its outgoing packets, but some older software products decline to respond to version 2 queries. The −**o** switch can be used to accommodate these products. Valid values for *version#* are 1, 2 or 3.

−**s** Log **ntpdate** actions via the **syslog**(3) facility rather than sending it to the standard output. This is a useful option when running the program from **cron**(1M).

−**a** *key#* Authenticate all packets using the key number indicated.

−**e** *authdelay*
Delay authentication by the specified number of seconds (see **xntpd**(1M) for

details). This number is usually small enough to be negligible for **ntpdate**'s purposes, though specifying a value may improve timekeeping on very slow CPU's.

–**k** *keyfile*
: Read the keys to be modified from the specified file rather than the default, **/opt/SUNWxntp/etc/ntp.keys**. *keyfile* should be in the format described in **xntpd**(1M).

–**p** *samples*
: Acquire the specified number of samples from each server, where *samples* is a number between 1 and 8, inclusive. The default is 4.

–**t** *timeout*
: Wait the specified amount of time for a response, rounded to the nearest 0.2 seconds. The default is 1 second, a value suitable for polling across a LAN.

FILES

/opt/SUNWxntp/etc/ntp.keys contains the encryption keys used by **ntpdate**.

SEE ALSO

xntpd(1M)

NAME

ntpq – standard ntp query program

SYNOPSIS

ntpq [–**inp**] [–**c** *command*] [*host*] [...]

DESCRIPTION

ntpq queries NTP servers that implement the recommended NTP mode 6 control message format about current state and requests changes in that state. It can also obtain and print a list of peers in a common format by sending multiple queries to the server.

ntpq can be run in interactive mode, or controlled via command line arguments. Requests to read and write arbitrary variables can be assembled, with raw and pretty–printed output options available.

If one or more request options is included on the command line when **ntpq** is executed, each of the requests is sent to the NTP servers running on each of the hosts given as command line arguments, or on *localhost* by default. If no request options are given, **ntpq** attempts to read commands from the standard input and execute them on the NTP server running on the first host given on the command line, again defaulting to *localhost* when no other host is specified. **ntpq** prompts for commands if the standard input is a terminal device.

ntpq uses NTP mode 6 packets to communicate with the NTP server, so it can be used to query any compatible server on the network that permits it.

NOTE: Since NTP is a UDP protocol, this communication is somewhat unreliable, especially over large distances in terms of network topology. **ntpq** makes one attempt to retransmit requests, and times-out requests if it does not hear from the remote host within a suitable time.

OPTIONS

Options other than –**i** and –**n** cause the specified queries to be sent to the indicated hosts immediately. Otherwise, **ntpq** attempts to read interactive format commands from the standard input.

–**c** *command*
: Interpret *command* as an interactive format command and add it to the list of commands to be executed on the specified host(s). Multiple –**c** options may be specified.

–**i**
: Operate in interactive mode. Prompts are written to the standard output and commands read from the standard input.

–**n**
: Output all host addresses in dotted–quad numeric format rather than converting to the canonical host names.

–**p**
: Print a list of the peers known to the server as well as a summary of their state. This is equivalent to the **peers** interactive command.

INTERNAL COMMANDS

Interactive commands consist of a keyword followed by zero to four arguments. Only enough characters of the full keyword to uniquely identify the command need be typed. The output of a command is normally sent to the standard output, but optionally, the output of individual commands can be sent to a file by appending a **>**, followed by a file name, to the command line.

A number of interactive format commands are executed entirely within the **ntpq** program itself and do not result in NTP mode 6 requests being sent to a server. They are:

? [*command_keyword*]
A **?** by itself prints a list of all the command keywords known to this incarnation of **ntpq**. A **?** followed by a command keyword prints function and usage information about the command. This command is probably a better source of information about **ntpq** than this manual page.

timeout *millseconds*
Allow the specified time out period for responses to server queries. The default is about 5000 milliseconds. Note that since **ntpq** retries each query once after a time out, the total waiting time for a time out is twice the time out value set.

delay *milliseconds*
Add the specified a time interval to timestamps included in requests that require authentication. This option enables (unreliable) server reconfiguration over long-delay network paths, or between machines whose clocks are unsynchronized. Actually, the server does not now require timestamps in authenticated requests, so this command may be obsolete.

host *hostname*
Send future queries to *hostname*. *hostname* may be either a host name or a numeric address.

poll [*#*] [**verbose**]
Poll the current server in client mode. The first argument is the number of times to poll (the default is 1) while the second argument may be given to obtain a more detailed output of the results. This command is currently just wishful thinking.

keyid *#*
Use the specified key number to authenticate configuration requests. # must correspond to a key number the server has been configured to use for this purpose.

passwd
Prompt the user to type in a password (which is not echoed) that will be used to authenticate configuration requests. The password must correspond to the key configured for use by the NTP server for this purpose.

hostnames yes|no
If **yes**, print host names in information displays; if **no**, print numeric addresses, instead. The default is **yes**, unless it was modified with **−n** on the command line.

raw Cause all output from query commands to print as received from the remote server. The only formating/interpretation done on the data is to transform non-ASCII data into a printable (but barely understandable) form.

cooked Cause output from query commands to be **cooked**. Variables that are recognized by the server will have their values reformatted for human consumption. Variables that **ntpq** thinks should have a value that can be decoded, but do not, are marked with a trailing **?**.

ntpversion 1|2|3
Set the NTP version number that **ntpq** claims in packets. The defaults is 3. Note that mode 6 control messages (and modes, for that matter) did not exist in NTP version 1. There appear to be no servers left that demand version 1.

authenticate yes|no
Send authentication on all requests. Normally, **ntpq** only authenticates write requests. Authenticated requests cause some servers to handle requests slightly differently, and can occasionally damage the CPU if authentication is turned on before a peer display is done.

addvars *variable_name*[=*value*] [,...]
rmvars *variable_name* [,...]
clearvars
The data carried by NTP mode 6 messages consists of a list of items of the form *variable_name*=*value*, where the =*value* is ignored, and can be omitted, in requests to the server to read variables. **ntpq** maintains an internal list in which data to be included in control messages can be assembled, and sent using the **readlist** and **writelist** commands described below. The **addvars** command allows variables and their optional values to be added to the list. If more than one variable is to be added, the list should be comma–separated and not contain white space. The **rmvars** command removes individual variables from the list, while **clearlist** removes all variables from the list.

debug more|less|off
Turn internal query program debugging on and off.

quit Exit **ntpq**.

CONTROL MESSAGE COMMANDS

Each peer known to an NTP server has a 16-bit integer *association identifier* assigned to it. NTP control messages that carry peer variables must identify the peer to which the values correspond by including its association ID. An association ID of 0 is special, and indicates the variables are system variables, whose names are drawn from a separate name space.

Control message commands result in one or more NTP mode 6 messages being sent to the server, and cause the data returned to be printed in some format. Most commands currently implemented send a single message and expect a single response. The current exceptions are the **peers** command, which sends a preprogrammed series of messages to obtain the data it needs, and the **mreadlist** and **mreadvar** commands, which iterate over a range of associations.

associations
Obtain and print a list of association identifiers and peer statuses for in–spec peers of the server being queried. The list is printed in columns. The first of these is an index numbering the associations from 1 for internal use, the second the actual association identifier returned by the server and the third the status word for the peer. This is followed by a number of columns containing data decoded from the status word. Note that the data returned by the **associations** command is cached internally in **ntpq**. The index is then of use when dealing with stupid servers which use association identifiers which are hard for humans to type, in that for any subsequent commands which require an association identifier as an argument, the form **&***index* may be used as an alternative.

lassociations
Obtain and print a list of association identifiers and peer statuses for all associations for which the server is maintaining state. This command differs from the **associations** command only for servers that retain state for out–of–spec client associations (*i.e.*, fuzzballs). Such associations are normally omitted from the display when the **associations** command is used, but are included in the output of **lassociations**.

passociations
Print association data concerning in–spec peers from the internally cached list of associations. This command performs identically to the **associations** except that it displays the internally stored data rather than making a new query.

lpassociations
Print data for all associations, including out–of–spec client associations, from the internally cached list of associations. This command differs from **passociations** only when dealing with fuzzballs.

pstatus *assocID*
Send a read status request to the server for the given association. The names and values of the peer variables returned will be printed. Note that the status word from the header is displayed preceding the variables, both in hexadecimal and in pidgin English.

readvar [*assocID*] [*variable_name*[**=***value*] [,...]]
Ask the server to return the values of the specified variables by sending a read variables request. If the association ID is omitted or is given as zero the variables are system variables; otherwise, they are peer variables and the values returned will be those of the corresponding peer. Omitting the variable list sends a request with no data, which should induce the server to return a default display.

rv [*assocID*] [*variable_name*[=*value*] [,...]]

An easy–to–type short form for the **readvar** command.

writevar *assocID variable_name*=*value* [,...]

Like the **readvar** request, except the specified variables are written instead of read.

readlist [*assocID*]

Ask the server to return the values of the variables in the internal variable list. If the association ID is omitted or is 0, the variables are assumed to be system variables. Otherwise, they are treated as peer variables. If the internal variable list is empty a request is sent without data, which should induce the remote server to return a default display.

rl [*assocID*]

An easy–to–type short form of the **readlist** command.

writelist [*assocID*]

Like the **readlist** request, except the internal list variables are written instead of read.

mreadvar *assocID assocID* [*variable_name*[=*value*] [,...]]

Like the **readvar** command except the query is done for each of a range of (nonzero) association IDs. This range is determined from the association list cached by the most recent **associations** command.

mrv *assocID assocID [variable_name*[=*value*] [,...]]

An easy–to–type short form of the **mreadvar** command.

mreadlist *assocID assocID*

Like the **readlist** command except the query is done for each of a range of (nonzero) association IDs. This range is determined from the association list cached by the most recent **associations** command.

mrl *assocID assocID*

An easy–to–type short form of the **mreadlist** command.

clockvar [*assocID*] [*variable_name*[=*value*] [,...]]

Send a list of the server's clock variables. Servers that have a radio clock or other external synchronization respond positively to this. If the association identifier is omitted or zero, the request is for the variables of the **system clock** and generally gets a positive response from all servers with a clock. If the server treats clocks as pseudo–peers, and hence can possibly have more than one clock connected at once, referencing the appropriate peer association ID shows the variables of a particular clock. Omitting the variable list causes the server to return a default variable display.

cv [*assocID*] [*variable_name*[=*value*] [,...]]

An easy–to–type short form of the **clockvar** command.

peers Obtain a list of in–spec peers of the server, along with a summary of each peer's state. Summary information includes the address of the remote peer, the reference ID (0.0.0.0 if the refID is unknown), the stratum of the remote peer, the polling interval in seconds, the reachability register in octal, and the current estimated delay, offset and dispersion of the peer, all in seconds. In addition, the character in the left margin indicates the fate of this peer in the clock selection algorithm. Characters only appear beside peers which were included in the final stage of the clock selection algorithm. A **.** indicates that this peer was cast off in the falseticker detection, while a **+** indicates that the peer made it through. A * denotes the peer the server is currently synchronizing with. Note that since the **peers** command depends on the ability to parse the values in the responses it gets, it may fail to work from time to time with servers that poorly control the data formats.

The contents of the host field may be one of four forms: host name, IP address, reference clock implementation name with its parameter, or **REFCLK(***implementation number*, *parameter*). On **hostnames no** only, IP–addresses are displayed.

lpeers Like **peers**, except a summary of all associations for which the server is maintaining state is printed. This can produce a much longer list of peers from fuzzball servers.

opeers An old form of the **peers** command with the reference ID replaced by the local interface address.

NOTES

The **peers** command is non–atomic and may occasionally result in spurious error messages about invalid associations occurring and terminating the command.

The timeout time is a fixed constant, which means you wait a long time for time outs since it assumes sort of a worst case. The program should improve the time out estimate as it sends queries to a particular host, but it doesn't.

NAME

ntptrace – trace ntp hosts back to their master time source

SYNOPSIS

ntptrace [**–vdn**] [**–r** *retries*] [**–t** *timeouts*] [*server*]

DESCRIPTION

ntptrace determines where a given Network Time Protocol (NTP) server gets its time and follows the chain of NTP servers back to their master time source. If given no arguments, it starts with ‘‘localhost.’’

Here is an example of the output from **ntptrace**:

```
% ntptrace
localhost: stratum 4, offset 0.0019529, synch distance 0.144135
server2.bozo.com: stratum 2, offset 0.0124263, synch distance 0.115784
usndh.edu: stratum 1, offset 0.0019298, synch distance 0.011994, refid
’WWVB’
```

On each line, the fields are (left to right): the host name, the host’s stratum, the time offset between that host and the local host (as measured by **ntptrace**; this is why it is not always zero for ‘‘localhost’’), the host’s ‘‘synchronization distance,’’ and (only for stratum-1 servers) the reference clock ID. All times are given in seconds. Synchronization distance is a measure of the goodness of the clock’s time.

OPTIONS

–d　Turn on some debugging output.

–n　Turn off the printing of host names, showing host IP addresses, instead. This may be necessary if a nameserver is down.

–r *retries*
　Set the number of retransmission attempts for each host to *retries* times. The default is 5.

–t *timeout*
　Set the retransmission timeout (in seconds) to *timeout*. The default is 2.

–v　Print verbose information about the NTP servers.

SEE ALSO

xntpd(1M), **xntpdc**(1M).

NOTES

This program makes no attempt to improve accuracy by doing multiple samples.

NAME

tickadj – fiddle time-related variables in the kernel

SYNOPSIS

tickadj [–**Ads**] [–**a** *new_tickadj*] [–**t** *new_tick*]

DESCRIPTION

The **tickadj** program uses **/dev/kmem** to read, and optionally modify, the following time–keeping–related variables in the running kernel: `tick`, which is the number of microseconds added to the system time during a clock interrupt; `tickadj`, which sets the slew rate and resolution used by the **adjtime**(2) system call; and `dosynctodr`, which indicates to the kernels on some machines whether they should internally adjust the system clock to keep it in line with with time–of–day clock.

By default, with no arguments, the **tickadj** program reads the variables of interest in the kernel and prints them. At the same time it determines an *optimal* value for the `tickadj` variable if the intent is to run the **xntpd**(1M) Network Time Protocol daemon, and prints this, as well. Since the operation of **tickadj** when reading the kernel mimics the operation of similar parts of the **xntpd**(1M) program fairly closely, this is useful for doing debugging of problems with **xntpd**(1M).

–**a** *new_tickadj*
: Set the variable `tickadj` to the value specified.

–**A**
: Set `tickadj` to the internally computed optimal value.

–**t** *new_tick*
: Reset the kernel's value of `tick`. This capability is useful on machines that have broken clocks.

–**s**
: Set the value of the variable `dosynctodr` to zero, which is a prerequisite for running the **xntpd**(1M) daemon under Solaris.

–**d**
: Report debug messages only. Normally, **tickadj** is quite verbose about what it is doing.

Use caution when running **tickadj** for the first time on a machine. The operations **tickadj** tries to perform are not guaranteed to work on all UNIX machines.

FILES

/unix, dev/kmem

SEE ALSO

xntpd(1M)

NAME

xntpd – network time protocol daemon

SYNOPSIS

xntpd [–**ab**] [–**c** *configfile*] [–**e** *authdelay*] [–**f** *driftfile*] [–**k** *keyfile*] [–**l** *loopfile*] [–**p** *pidfile*] [–**r** *broaddelay*] [–**s** *statsdir*] [–**t** *trustedkey*]

DESCRIPTION

xntpd is a daemon that maintains a UNIX system's time–of–day in agreement with Internet standard time servers. It is a complete implementation of the Network Time Protocol (NTP) Version 2 standard as defined by RFC 1119, and also retains compatibility with Version 1 servers as defined by RFC 1059.

xntpd does all computations in fixed-point arithmetic and is entirely free of floating point code. The computations done in the protocol and clock adjustment code are carried out with high precision and with attention to details that might introduce systematic bias into the integrations, to try to maintain an accuracy suitable for synchronizing with even the most precise external time source.

Ordinarily, **xntpd** reads its configuration from a file at startup time. The default configuration file is **/opt/SUNWxntp/etc/ntp.conf,** though this may be overridden from the command line. It is also possible to specify a working, though limited, **xntpd** configuration entirely on the command line, obviating the need for a configuration file. This may be particularly appropriate when **xntpd** is to be configured as a broadcast client, with all peers determined by the listening to broadcasts at runtime. Various internal **xntpd** variables can be displayed, and configuration options altered, while the daemon is running through use of the **xntpdc**(1M) program.

OPTIONS

-a Run in authenticate mode.

-b Listen for broadcast NTP and, if it is available, sync to it.

-c *configfile*
Use *configfile* as the configuration file.

-e *authdelay*
Take the specified number of seconds to compute the NTP encryption field on this computer.

-f *driftfile*
Assume the drift file is located at location *driftfile.*

-k *keyfile*
Assume that *keyfile* contains the NTP authentication keys.

-l *loopfile*
Record loop filter statistics in the specified file.

-p *pidfile*
Record the daemon's process ID in the specified file.

-r *broaddelay*
Use the specified time as the default round trip, in seconds, to be used when synchronizing to broadcasts.

-s *statsdir*
Use the specified directory for creating statistics files.

-t *trustedkey*
Add the specified key number to the trusted key list.

CONFIGURATION FILE OPTIONS

xntpd's configuration file is relatively free format. Comments, which may be freely inserted, begin with a **#** character and extend to the end of the line. Blank lines are ignored. Configuration statements include an initial keyword followed by white-space-separated arguments, some of which may be optional. Configuration statements may not be continued over multiple lines. Arguments may be network numbers (which must be written in numeric, dotted–quad form), integers, floating point numbers (when specifying times in seconds) and text strings.

peer *host_address* [**key** *#*] [**version** *#*] [**minpoll**] [**prefer**]

server *host_address* [**key** *#*] [**version** *#*] [**minpoll**] [**prefer**]

broadcast *host_address* [**key** *#*] [**version** *#*] [**minpoll]**

The three statements above specify various time servers to be used or provided. The **peer** statement specifies that the given host is to be polled in **symmetric active** mode, *i.e.*, that the host is requested to provide time to which you might synchronize). Also, it indicates that you are willing to synchronize a remote host to your time if need be. The **server** statement specifies that the given host is to be polled in **client** mode, that is, that the host is requested to provide time that you might synchronize with but that you are unwilling to have the remote host synchronize to your own time. The **broadcast** statement requests your local daemon to transmit broadcast NTP to the specified address. The latter is usually the broadcast address on at least one of your local networks.

The options are:

key *#* For all packets sent to the address, include authentication fields encrypted via the specified key number (the range of which is that of an unsigned 32-bit integer). The default is to not include an encryption field.

version *#*"
Use the specified version number for outgoing NTP packets. Versions 1, 2, and 3 are the choices; Version 3 is the default.

minpoll
Keep the polling interval clamped at the minimum (64 seconds), even when the local daemon isn't using the remote server's data for synchronization. This is the default for broadcasting; otherwise, this option should not be specified, except for testing, since it increases traffic to the servers.

prefer Mark the host as a preferred host. Preferred hosts determine the validity of the PPS signal and are the primary selection for synchronization when found in the set of suitable synchronization sources.

precision *#*

Use the specified precision for local timekeeping. The value is an integer that is approximately the base 2 logarithm of the local time-keeping precision, in seconds. By default this value is set to -6.

The precision declared by an implementation can affect several aspects of server operation, and can be used as a tuning parameter for your synchronization subnet. It should probably not be changed from the default value, however, unless there is a good reason to do so.

driftfile *filename*

The name of the file used to record the **drift** (or frequency error) value computed by **xntpd**. If the file exists on startup, it is read and the value used to initialize **xntpd**'s internal value of the frequency error. The file is then updated once every hour by replacing the old file with a new one containing the current value of the frequency error. Note that the file is updated by first writing the current drift value into a temporary file and then using **rename**(2) to replace the old version. This implies that **xntpd** must have write permission for the directory in which the drift file is located, and that file system links, symbolic or otherwise, should probably be avoided.

monitor yes | no

Enable / disable the **xntpd** traffic-monitoring function. When enabled, the origin address of each packet received by the server is recorded along with a limited amount of additional information, such as the mode of the request and whether it originated from an NTP server port. Inspect traffic monitoring data via the **xntpdc**(1M) command **monlist**. The default is **no**, *i.e.*, traffic monitoring should not be done.

Note that the traffic monitoring facility increases the CPU time used by **xntpd**, as well as increasing the daemon's memory utilization by as much as 8.5 kilobytes. This facility is normally useful for the detection of peers with malfunctioning software or which are sending bogus data. It is primarily intended for very popular servers which exchange time with large numbers of peers, though it may also be useful for access monitoring of local servers if you are willing to accept the overhead.

broadcastclient yes | no

Make the local server listen for and attempt to synchronize to broadcast NTP – or do not. The default is **no**.

broadcastdelay *seconds*

Use the specified time as the round trip delay to the host to whom broadcasts are being synchronized. The value is specified in seconds and is typically (for Ethernet) a number between 0.007 and 0.015 seconds. This initial estimate may be improved by the polling of each server to determine a more accurate value. The default is 0.008 seconds.

authenticate yes | no

Operate, or do not operate, in authenticate mode. If **yes**, only peers that include an authentication field encrypted with one of our trusted keys (see below) are considered as candidates for synchronizing to. The default is **no**.

authdelay *seconds*

Use the time shown as that necessary to encrypt an NTP authentication field on the local computer. This value is used to correct transmit timestamps when the authentication is used on outgoing packets. The value usually lies somewhere in the range 0.0001 seconds to 0.003 seconds, though is very dependent on the CPU speed of the host computer. The value is usually computed using the **authspeed** program included with the distribution.

keys *filename*

Use the encryption keys in *filename* as those to be used by **xntpd**. The format of this file is described below.

trustedkey *#* [*#* ...]

Use the specified encryption key numbers trusted for the purposes of determining peers suitable for time synchronization, when authentication is enabled. Only peers using one of these keys for encryption of the authentication field, and whose authenticity can be verified by successful decryption, are considered as synchronization candidates. The arguments are 32-bit unsigned integers. Note, however, that NTP key 0 is fixed and globally known. If meaningful authentication is to be performed, the 0 key should not be trusted.

requestkey *#*

xntpd allows runtime reconfiguration to be performed via the **xntpdc**(1M) program. Such requests must be authenticated. The **requestkey** statement allows the specification of a 32-bit unsigned integer key number to be used for authenticating such requests. Note that if no **requestkey** statement is included in the configuration file, the runtime reconfiguration facility is disabled.

controlkey *#*

Certain changes can be made to the **xntpd** server via mode 6 control messages, in particular, the setting of leap-second indications in a server with a radio clock. The **controlkey** statement specifies an encryption key number to be used for authenticating such messages. Omitting this statement causes control messages that would change the state of the server to be ignored.

restrict *address* [**mask** *numeric_mask*] [*flag*] [...]

xntpd implements a general-purpose address–and–mask based restriction list. The list is sorted by address and by mask, and is searched in this order for matches, with the last match found defining the restriction flags associated with the incoming packets. The source address of incoming packets is used for the match, with the 32-bit address being and'ed with the mask associated with the restriction entry and then compared with the entry's address (which has also been and'ed with the mask) to look for a match. The *mask* argument defaults to 255.255.255.255, meaning that the *address* is treated as the address of an individual host. A default entry (address 0.0.0.0, mask 0.0.0.0) is always included and, given the sort algorithm, is always the first entry in the list. Note that, while *address* is normally given as a dotted–quad address, the text string **default**, with no mask option, may be used to indicate the default entry.

In the current implementation flags always restrict access; *i.e.*, an entry with no flags indicates that free access to the server is to be given. The flags are not orthogonal, in that more restrictive flags often make less restrictive ones redundant. The flags can generally be classed into two categories, those that restrict time service and those that restrict informational queries and attempt to do runtime reconfiguration of the server. One or more of the following flags may be specified:

ignore Ignore all packets from hosts that match this entry. If this flag is specified, neither queries nor time server polls are responded to.

noquery
Ignore all NTP mode 6 and 7 packets (*i.e.*, information queries and configuration requests) from the source. Time service is not affected.

nomodify
Ignore all NTP mode 6 and 7 packets that attempt to modify the state of the server (*i.e.*, runtime reconfiguration). Queries that return information are permitted.

notrap Decline to provide mode 6 control message trap service to matching hosts. The trap service is a subsystem of the mode 6 control message protocol, which is intended for use by remote event logging programs.

lowpriotrap
Declare traps set by matching hosts to be low priority. The number of traps a server can maintain is limited (the current limit is 3). Traps are usually assigned on a first come, first served basis, with later trap requesters being denied service. This flag modifies the assignment algorithm by allowing low priority traps to be overridden by later requests for normal priority traps.

noserve
Ignore NTP packets whose mode is other than 6 or 7. In effect, time service is denied, though queries may still be permitted.

nopeer Provide stateless time service to polling hosts, but do not allocate peer memory resources to these hosts even if they otherwise might be considered useful as future synchronization partners.

notrust Treat these hosts normally in other respects, but never use them as synchronization sources.

ntpport
This is actually a match algorithm modifier, rather than a restriction flag. Its presence causes the restriction entry to be matched only if the source port in the packet is the standard NTP UDP port (123). Both **ntpport** and non–**ntpport** may be specified. The **ntpport** is considered more specific and is sorted later in the list.

Default restriction list entries with the flags **ignore, ntpport** for each of the local host's interface addresses are inserted into the table at startup to prevent the server from attempting to synchronize to its own time. A default entry is also always present, though if it is otherwise unconfigured, no flags are associated with the default entry (*i.e.*, everything besides your own NTP server is unrestricted).

The restriction facility was added to allow the current access policies of the time servers running on the NSFnet backbone to be implemented with **xntpd**, as well. While this facility may be otherwise useful for keeping unwanted or broken remote time servers from affecting your own, it should not be considered an alternative to the standard NTP authentication facility. Source address based restrictions are easily circumvented by a determined cracker.

trap *host_address* [**port** *port_number*] [**interface** *interface_address*]

Configure a trap receiver at the given host address and port number, sending messages with the specified local interface address. If the port number is unspecified, a value of 18447 is used. If the interface address is not specified, the message is sent with a source address that is that of the local interface through which the message is sent. Note that on a multihomed host the interface used may vary from time to time with routing changes.

The trap receiver generally logs event messages and other information from the server in a log file. While such monitor programs may also request their own trap dynamically, configuring a trap receiver ensures that no messages are lost when the server is started.

maxskew *seconds*

Set the system maximum skew parameter to the number of seconds given. The default value is 0.010 seconds. This is a tuning parameter of use in improving performance when network link conditions are poor, and should probably not be changed unless your server is to run under exceptional conditions.

select *algorithm_number*

Use the specified (one of five) selection weight algorithms. The default is algorithm number 1, which is the algorithm specified in RFC 1119. Algorithm numbers 2 through 5 select alternative, experimental selection weighting algorithms, all of which tend to give a greater degree of trust to either lower stratum and / or lower delay peers than the standard algorithm.

resolver */path/xntpres*

Make the daemon use the specified path to the **xntpres** program. This utility is used when names that require resolution (rather than numeric addresses) are used in **peer** and **server** entries in the configuration file. As **xntpres** makes use of mode 7 runtime reconfiguration, this facility must also be enabled if the procedure is to succeed (see the **requestkey** and **keys** statements above).

statsdir */directory*

Use the specified full path and directory as the place where statistics files should be created (see below). This keyword allows the (otherwise constant) **filegen** filename prefix to be modified for file generation sets used for handling statistics logs (see the **filegen** statement below).

statistics *name...*

Enable writing of statistics records. Currently, two kinds of statistics are supported, **loopstats** and **peerstats**.

loopstats enables recording of loop filter statistics information. Each computation of the local clock parameters outputs a line of the following form to the file generation set named **loopstats**:

```
48773 10847.650 0.0001307 17.3478 2
```

The first two fields show the date (modified Julian) and time (seconds and fraction past UTC midnight). The next three fields show last offset, current drift compensation value, and time constant of the loop filter.

peerstats enables recording of peer statistics information, which include statistics records of all peers of a NTP server and of the PPS filter, if PPS signal handling is supported by the server. Each valid update appends a line of the following form to the current element of a file generation set named **peerstats**:

```
48773 10847.650 127.127.4.1 9714 -0.001605 0.00000 0.00142
```

The first two fields show the date (modified Julian) and time (seconds and fraction past UTC midnight), while the next two fields are the peer address and status, respectively. The final three fields show offset, delay and dispersion.

Statistic files are managed via file generation sets (see **filegen** below). The information obtained by the enabling of statistics recording allows analysis of temporal properties of a **xntpd** server. It is usually only useful to primary servers or, perhaps, main campus servers.

filegen *name* [**file** *filename*] [**type** *typename*] [**flag** *flagval*] [**link** **nolink**] [**enable** **disable**]

Configure the setting of generation file set *name*. Generation file sets provide a means for handling files that are continuously growing during the lifetime of a server. Server statistics are a typical example for such files. Generation file sets provide access to a set of files used to store the actual data. At any time at most one element of the set is being written to. The *type* given specifies when and how data will be directed to a new element of the set. This way, information stored in elements of a file set that are currently unused are available for administrational operations without the risc of disturbing the operation of **xntpd**. **NOTE:** They can be removed to free space for new data produced.

Filenames of set members are built from three elements.

prefix This is a constant filename path. It is not subject to modifications via the **filegen** statement. It is defined by the server, usually specified as a compile time constant. It may, however, be configurable for individual file generation sets via other commands. For example, the prefix used with **loopstats** and **peerstats** filegens can be configured using the **statsdir** statement explained above.

filename
This string is directly concatenated to the *prefix* mentioned above (no intervening **/** (slash)). This can be modified via the **file** argument to the **filegen** statement. No **..** elements are allowed in this component to prevent filenames from referring to parts outside the file system hierarchy denoted by **prefix**.

suffix This part reflects individual elements of a file set. It is generated according to the *type* of a file set as explained below.

A file generation set is characterized by its type. The following types are supported:

none The file set is actually a single plain file.

pid One element of file set is used per incarnation of a **xntpd** server. This type does not perform any changes to file set members during runtime; however, it provides an easy way of separating files belonging to different **xntpd** server incarnations.

The set member filename is built by the appending of a dot (**.**) to concatenated **prefix** and **filename** strings, and the appending of the decimal representation of the process ID of the **xntpd** server process.

day One file generation set element is created per day. The term **day** is based on **UTC**. A day is defined as the period between 00:00 and 24:00 UTC. The file set member suffix consists of a dot **.** and a day specification in the form < *YYYYMMDD* >, where *YYYY* is a four-digit year number (e.g. 1994); *MM* is a two-digit month number, *DD* is a two-digit day number. Thus, all information written at December 10th, 1994 would end up in a file named *prefix filename***.19941210**.

week Any file set member contains data related to a certain week of a year. To define the term **week** compute **day of year** modulo 7. Elements of such a file generation set are distinguished by the appending of the following suffix to the file set filename base: A dot, a four-digit year number, the letter **W**, and a two-digit week number. For example, information from January, 10th 1992 would end up in a file with suffix **.1992W1**.

month One generation file set element is generated per month. The file name suffix consists of a dot, a four-digit year number, and a two-digit month.

year One generation file elment is generated per year. The filename suffix consists of a dot and a four-digit year number.

age This type of file generation set changes to a new element of the file set every 24 hours of server operation. The filename suffix consists of a dot, the letter **a**, and an eight-digit number. This number is taken to be the number of seconds the server is running at the start of the corresponding 24 hour period.

Information is only written to a file generation set when this set is **enabled**. Output is prevented by specification of **disabled**.

It is convenient to be able to access the *current* element of a file generation set by a fixed name. To enable this feature, specify **link**; to disable it, use **nolink**. If **link** is specified, a hard link from the current file set element to a file without suffix is created. When there is already a file with this name and the number of links of this file is one, it is renamed appending a dot, the letter **C**, and the pid of the **xntpd** server process. When the number of links is greater than one, the file is unlinked. This allows the current file to be accessed by a constant name.

AUTHENTICATION KEY FILE FORMAT

The NTP standard specifies an extension that allows verification of the authenticity of received NTP packets, and to provide an indication of authenticity in outgoing packets. This is implemented in **xntpd** using the DES encryption algorithm. (**NOTE:** the **xntpd** daemon provided by Sun Microsystems does not include the DES encryption algorithm and its capabilities. The MD5 encryption algorithm is provided.)

The DES specification allows any one of a possible 4 billion keys, numbered with 32-bit unsigned integers, to be used to authenticate an association. The servers involved in an association must agree on the value of the key used to authenticate their data, though they must each learn the key independently. The keys are standard 56-bit DES keys.

A new experimental authentication algorithm is also available that uses an MD5 message digest to compute an authenticator. Currently the length of the key or password is limited to 8 characters, but this will eventually be changed to accommodate an effectively unlimited password phrase. **xntpd** reads its keys from a file specified via the **-k** command line option or the **keys** statement in the configuration file. While key number 0 is fixed by the NTP standard (as 56 zero bits) and may not be changed, one or more of the keys numbered 1 through 15 may be arbitrarily set in the keys file.

The key file uses the same comment conventions as the configuration file. Key entries use a fixed format of the form:

keyno type key

where *keyno* is a positive integer, *type* is a single character that defines the format the key is given in, and *key* is the key itself.

The key may be given in one of three different formats, controlled by the *type* character. The three key types, and corresponding formats, are:

S The *key* is a 64-bit hexadecimal number in the format specified in the DES document; that is, the high order 7 bits of each octet are used to form the 56-bit key while the low order bit of each octet is given a value such that odd parity is maintained for the octet. Leading zeroes must be specified (*i.e.*, the key must be exactly 16 hex digits long) and odd parity must be maintained. Hence a zero key, in standard format, would be given as **0101010101010101**.

N The *key* is a 64-bit hexadecimal number in the format specified in the NTP standard. This is the same as the DES format except the bits in each octet have been rotated one bit right so that the parity bit is now the high order bit of the octet. Leading zeroes must be specified and odd parity must be maintained. A zero key in NTP format would be specified as **8080808080808080**

A The *key* is a 1–to–8 character ASCII string. A key is formed from this by using the lower order 7 bits of the ASCII representation of each character in the string, with zeroes being added on the right when necessary to form a full width 56-bit key, in the same way that encryption keys are formed from UNIX passwords.

M The *key* is a 1–to–8 character ASCII string, using the MD5 authentication scheme. Note that both the keys and the authentication schemes (DES or MD5) must be identical between a set of peers sharing the same key number.

One of the keys may be chosen, by way of the configuration file **requestkey** statement, to authenticate runtime configuration requests made via the **xntpdc**(1M) program. The latter program obtains the key from the terminal as a password, so it is generally appropriate to specify the key chosen to be used for this purpose in ASCII format.

PRIMARY CLOCK SUPPORT

xntpd can be optionally compiled to include support for a number of types of reference clocks. A reference clock is generally (though not always) a radio timecode receiver that is synchronized to a source of standard time such as the services offered by the NRC in Canada, and NIST in the U.S. The interface between the computer and the timecode receiver is device-dependent and varies, but is often a serial port.

For the purposes of configuration, **xntpd** treats reference clocks in a manner analogous to normal NTP peers as much as possible. Reference clocks are referred to by address, much as a normal peer is, though an invalid IP address is used to distinguish them from normal peers. Reference clock addresses are of the form **127.127.**_t.u_, where _t_ is an integer denoting the clock type and _u_ indicates the type–specific unit number. Reference clocks are normally enabled by the configuration of the clock as a server using a **server** statement in the configuration file that references the clock's address (configuring a reference clock with a **peer** statement can also be done, though with some clock drivers this may cause the clock to be treated somewhat differently and by convention is used for debugging purposes). Clock addresses may generally be used anywhere else in the configuration file a normal IP address can be used, for example in **restrict** statements.

There is one additional configuration statement that becomes valid when reference clock support has been compiled in. Its format is:

fudge _127.127.t.u_ [**time1** _secs_] [**time2** _secs_] [**value1** _int_] [**value2** _int_] [**flag1 0 | 1**] [**flag2 0 | 1**]

There are two times (whose values are specified in fixed point seconds), two integral values and two binary flags available for customizing the operation of a clock. The interpretation of these values, and whether they are used at all, is a function of the needs of the particular clock driver.

xntpd on UNIX machines currently supports several different types of clock hardware plus a special pseudo–clock used for backup or when no other clock source is available. The clock drivers, and the addresses used to configure them, are described following:

127.127.1.u

Local synchronization clock driver. This driver doesn't support an actual clock, but rather allows the server to synchronize to its own clock, in essence to free run without its stratum increasing to infinity. This can be used to run an isolated NTP synchronization network where no standard time source is available, by allowing a free running clock to appear as if it has external synchronization to other servers. By running the local clock at an elevated stratum it can also be used to prevent a server's stratum from rising above a fixed value, this allowing a synchronization subnet to synchronize to a single local server for periods when connectivity to the primary servers is lost.

The unit number of the clock (the least significant octet in the address) must lie in the range 0 through 15 inclusive and is used as the stratum the local clock will run at. Note that the server, when synchronized to the local clock, will advertise a stratum one greater than the clock peer's stratum. More than one local clock may be configured (indeed all 16 units may be active at once), though this hardly seems useful.

The local clock driver uses only the **fudge time1** parameter. This parameter actually provides read and write access to the local clock drift compensation register. This value, which actually provides a fine resolution speed adjustment for the local clock, is settable but will remain unchanged from any set value when the clock is free running without external synchronization. The **fudge time1** parameter thus provides a way manually adjust the speed of the clock to maintain reasonable synchronization with, say, a voice time announcement. It is actually more useful to manipulate this value with the **xntpdc**(1M) program.

127.127.3.u

Precision Standard Time 1010/1020 WWV/H Receiver. This driver can be used to receive time from a PST 1020 WWV receiver connected to a serial port on the computer. Any or all of four units, with unit numbers in the range 0 through 3, can be configured. The driver assumes the serial line the clock is connected to is **/dev/pst***%d* (*i.e.,* unit 1, at 127.127.3.1, opens **/dev/pst1** to speak to the clock) and that the PST receiver is configured for 9600 baud operation.

The **fudge time1** and **time2** parameters are used by the driver as nominal propagation delays when synchronized to WWV and WWVH, respectively. They default to 0.0075 and 0.0265 seconds, values that are about right for Toronto. While this should in principle be set to the propagation delays appropriate for the location, one should not feel inhibited from tweaking the values to make the output of the clock match other stratum 1 NTP peers more exactly. The only consideration when adjusting these is that the difference between the values should be kept set to the predicted difference in propagation delay between WWV and WWVH at all times.

The **value1** parameter can be used to set the stratum at which the peer operates. The default is 0, which is correct if you want the clock to be considered for synchronization whenever it is operating, though higher values may be assigned if you only want the clock to provide backup service when all other primary sources have failed. The **value2** parameter is set to the number of minutes that the daemon will allow the clock to go without synchronization before it starts disbelieving it. The default is 20, which is suitable if you have good quality backup NTP peers. If your network is isolated or your network connections are poor it might be advantageous to increase this value substantially.

The **fudge flag1** can be set with good effect if your system clock's precision is worse than about 500 microseconds, as it causes the code processing algorithms to be modified slightly in a way that should produce better results with imprecise clocks. Setting **fudge flag2** forces the driver to send to the clock the commands required to program the current WWV and WWVH fudge delays into it (this is normally done only when the values change). Setting the (otherwise undocumented) **fudge flag3** causes the driver to reset the clock. The latter two flags are generally only useful for debugging.

127.127.4.u

Spectracom 8170 and Netclock/2 WWVB Synchronized Clocks. This driver provides an interface to the Spectracom 8170 and Netclock/2 WWVB Synchronized Clocks in either Format-0 or Format-1 mode of operation at 9600 bps. The driver opens the RS232 output on **/dev/wwvb***%d*, where *%d* is replaced by the unit number of the unit opened. This driver does not require a 1-pulse-per-second (pps) signal and automatically compensates for the baud rate on the serial port. It does not require the clock discipline or STREAMs modules.

The **fudge time1** parameter is used as a general calibration factor for the clock, with positive values advancing the time and negative values retarding it. The parameter defaults to zero, which should be appropriate if the clock's propagation delay switches have been set appropriately. The value1 parameter can be used to set the stratum at which the peer operates. The default is 0, which is correct if you want the clock to be considered for synchronization whenever it is operating.

127.127.5.u

Kinemetrics TrueTime 468-DC Receiver. This driver provides an interface to a Kinemetrics TrueTime 468-DC GOES receiver. It opens **/dev/goes***%d* to listen to the clock, where *%d* is the unit number of the clock to be opened.

Since the clock adjusts to compensate for radio delays, no fudging is usually needed. If you know the internal delay DIP switches are wrong for your location (*i.e.*, set to factory default 50 when another setting is needed), you can manually advance / retard the clock with **fudge time1**.

While untested, this driver should work with all TrueTime receivers, since all use the same timecode.

127.127.7.u

Direct synchronization to the CHU timecode. Unlike the NIST time services, whose timecode requires quite specialized hardware to interpret, the CHU timecode can be received directly via a serial port after demodulation. While there are currently no commercial CHU receivers the hardware required to receive the CHU timecode is fairly simple to build. The driver supports four units, numbered 0 through 3, and opens **/dev/chu***%d* to access the serial device the particular unit is connected to. While it is possible to configure several CHU units simultaneously, this is not recommended as the character interrupts from all units will be occurring at the same time and will interfere with each other. Note that the operation of the CHU driver (and probably the ability to actually compile the driver into the daemon) requires that a special purpose CHU serial line discipline be installed in your kernel to take timestamps at interrupt level and to do noise prefiltering.

The CHU driver uses fudge time1 and time2 as calibration factors. By convention, time1 is the estimated propagation delay to CHU (the *propdelay* program in the distribution may be used to compute a rough estimate of this) while time2 is an estimate of propagation–independent delays through the modem filters and serial driver. These values default to 0.0025 and 0.0002 seconds, respectively, values that are suitable for Toronto and the modem I built. Fudge value1 can be set to the stratum you wish the clock peer to operate at. This defaults to zero, though a higher stratum can be set if the clock is only to be used for backup. If fudge flag1 is set a slightly different algorithm, better suited for machines where the system clock precision is coarser than about 500 microseconds, is selected for processing the raw data.

127.127.8.u

Synchronization to DCF receivers. The timecode of *DCF77* receivers is sampled via a STREAMS module in the kernel (the STREAMS module has been designed for use with SUN Systems under Solaris. It can be linked directly into the kernel or loaded via the loadable driver mechanism.) This STREAMS module can be adepted to be able to convert different time code formats. If the daemon is compiled without the STREAM definition, synchronization works without the Sun streams module, though accuracy is significantly degraded.

The actual receiver status is mapped into various synchronization states as used by Meinberg receivers. The STREAMS module is configured to interpret the time code of Meinberg DCF U/A 31 and PZF 535 receivers and others (see list below).

The reference clock support in **xntp** contains the necessary configuration tables for those receivers. In addition to supporting up to 8 different clock types and 16 devices, the generation of a PPS signal is also provided as a configuration option. The PPS configuration option uses the receiver-generated time stamps for feeding the PPS loopfilter control for much finer clock synchronization.

CAUTION: The PPS configuration option is different from the hardware PPS signal, which is also supported (see below), as it controls the way **xntpd** is synchronized to the reference clock, while the hardware PPS signal controls the way time offsets are determined.

The use of the PPS option requires receivers with an accuracy of better than 1ms.

FUDGE FACTORS

Only two fudge factors are utilized. The **time1** fudge factor defines the phase offset of the last-transmitted character to the actual time. The **flag0** enables input filtering. This is a median filter with continuous sampling. The **flag1** selects averaging of the samples remaining after the filtering. Leap second handling is controlled via **flag2.** When set, a leap second is deleted on receipt of a leap second indication from DCF77; otherwise, the leap second is added (which is the default).

ntpq TIMECODE VARIABLE

The timecode variable in the **ntpq read clock variable** command contains several fields. The first field is the local time in UNIX format. The second field is the offset to UTC (format *HHMM*). The currently active receiver flags are listed next. Additional feature flags of the receiver are optionally listed in parenthesis. The actual time code is enclosed in angle brackets < >. A qualification of the decoded time code format is following the time code. The last piece of information is the overall running time and the accumulated times for the clock event states.

UNIT ENCODING

The unit field <u> encodes the device, clock type and the PPS generation option. There are 16 possible devices that are encoded in the lower 4 bits of the <u> field. The devices are named **/dev/dcf77-0** through **/dev/dcf77-15**. Bits 4 through 6 encode the clock type. The fudge factors of the clock type are take from a table *clockinfo* in **refclock_dcf77.c**. The generation of PPS information for disciplining the local NTP clock is encoded in bit 7 of

<u>.

Currently, five clock types are supported:

0 Meinberg PZF535 receiver (FM demodulation / TCXO / 50us)

1 Meinberg PZF535 receiver (FM demodulation / OCXO / 50us)

2 Meinberg DCF U / A 31 receiver (AM demodulation / 6ms)

3 ELV DCF7000 (sloppy AM demodulation / 50ms)

The DCF77 reference clock support carefully monitors the state transitions of the receiver. All state changes and exceptional events such as loss of time code transmission are logged via the **syslog** facility. Every hour a summary of the accumulated times for the clock states is listed via **syslog**.

PPS support is only available when the receiver is completely synchronized. The receiver is believed to deliver correct time for an additional period of time after losing synchronization unless a disruption in time code transmission is detected (possible power loss). The trust period is dependent on the receiver oscillator and thus a function of clock type. This is one of the parameters in the *clockinfo* field of the reference clock implementation. This parameter cannot be configured by **xntpdc**.

In addition to the PPS loopfilter control, a true PPS hardware signal can be applied on Sun SPARCstations via the CPU serial ports on the CD pin. This signal is automatically detected and used for offset calculation. The input signal must be the time mark for the following time code. (The edge sensitivity can be selected – look into **dcf77sync.c** for details). Meinberg receivers can be connected by feeding the PPS pulse of the receiver via a 1488 level converter to Pin 8 (CD) of a Sun serial zs–port.

There exists a special firmware release for the PZF535 Meinberg receivers. This release (PZFUERL 4.6 or higher) is absolutely recommended for XNTP use, as it provides LEAP warning, time code / time zone information, and alternate antenna indication. Please check with Meinberg for this firmware release.

FILES

/opt/SUNWxntp/etc/ntp.conf
the default name of the configuration file

/opt/SUNWxntp/etc/ntp.drift
the conventional name of the drift file

/opt/SUNWxntp/etc/ntp.keys
the conventional name of the key file

SEE ALSO

ntpdate(1M), **ntpq**(1M), **xntpdc**(1M).

NOTES

xntpd has become larger than might be desirable for an elevated–priority daemon running on a workstation, particularly since many of the fancy features that consume the space were designed more with a busy primary server, rather than a high-stratum workstation, in mind. This will eventually be corrected either by adopting the **ntpd** daemon as an alternative when it becomes able to match **xntpd**'s performance, or if not, than by producing a stripped down version of **xntpd** specifically for workstation use.

NAME

xntpdc – query/control program for the ntp daemon

SYNOPSIS

xntpdc [–**ilnps**] [–**c** *command*] [–**i** *host*] [...]

DESCRIPTION

xntpdc queries the **xntpd**(1M) daemon about its current state and requests changes in that state. The program may be run in interactive mode, or controlled via command line arguments. Extensive state and statistics information is available through the **xntpdc** interface. In addition, nearly all the configuration options that can be specified at start up via the configuration file of **xntpd**(1M) may also be specified at run time via **xntpdc**.

If one or more request options is included on the command line when **xntpdc** is executed, each of the requests are sent to the NTP servers running on each of the hosts given as command line arguments, or on *localhost* by default. If no request options are given, **xntpdc** attempts to read commands from the standard input and execute them on the NTP server running on the first host given on the command line, again defaulting to *localhost* when no other host is specified. **xntpdc** prompts for commands if the standard input is a terminal device.

xntpdc uses NTP mode 7 packets to communicate with the NTP server, and hence, can be used to query any compatible server on the network that permits it. Note that since NTP is a UDP protocol this communication is somewhat unreliable, especially over large distances in terms of network topology. **xntpdc** makes no attempt to retransmit requests, and times-out requests if it does not hear from the remote host is not heard within a suitable time.

Command line options follow. Specifying a command line option other than –**i** or –**n** causes the specified queries to be sent to the indicated hosts immediately. Otherwise, **xntpdc** attempts to read interactive format commands from the standard input.

–**c** *command*
: Interpret *command* as an interactive command and add it to the list of commands to be executed on the specified hosts. Multiple –**c** options may be given.

–**i** *host*
: Force **xntpdc** to operate in interactive mode. Prompts are written to the standard output and commands read from the standard input.

–**l**
: Obtain a list of peers known to the servers. This switch is equivalent to –**c listpeers**.

–**n**
: Output all host addresses in dotted–quad numeric format rather than converting them to the canonical host names.

–**p**
: Print a list of the peers known to the server as well as a summary of their state. This is equivalent to –**c peers**.

–**s**
: Print a list of the peers known to the server as well as a summary of their state, but in a slightly different format than the –**p** switch. This is equivalent to **-c dmpeers**.

INTERNAL COMMANDS

Interactive format commands consist of a keyword followed by zero to four arguments. Only enough characters of the full keyword to uniquely identify the command need be typed. The output of a command is normally sent to the standard output; to send output of individual commands to a file, append a >, followed by a file name, to the command line.

A number of interactive format commands are executed entirely within the **xntpdc** program itself and do not result in NTP mode 7 requests being sent to a server. These are:

? [*command_keyword*]
: A **?** by itself prints a list of all the command keywords known to this incarnation of **xntpdc**. A **?** followed by a command keyword prints function and usage information about the command. This command is probably a better source of information about **xntpdc** than this manual page.

help [*command_keyword*]
: A synonym for the **?** command.

timeout *milliseconds*
: Use the specified time out period for responses to server queries. The default is about 8000 milliseconds.

delay *milliseconds*
: Use the specified time interval as that to be added to timestamps included in requests that require authentication. This is used to enable unreliable server reconfiguration over long delay network paths or between machines whose clocks are unsynchronized.

host *hostname*
: Set the host to which future queries will be sent. *hostname* may be either a host name or a numeric address.

poll [*#*] [**verbose**]
: Poll the current server in client mode. The first argument is the number of times to poll (the default is 1) while the second argument may be given to obtain a more detailed output of the results. This command is currently just wishful thinking.

keyid *#*
: Use the specified key number to authenticate configuration requests. It must correspond to the key number the server has been configured to use for this purpose.

passwd
: Prompt the user to type in a password (not echoed) that is used to authenticate configuration requests. The password must correspond to the key configured for use by the NTP server for this purpose.

hostnames yes | no
: If **yes** is specified, print host names in information displays; if **no** is given, print numeric addresses, instead. The default is **yes** unless it was modified via the command line –**n** switch.

quit Exit **xntpdc**.

QUERY COMMANDS

Query commands result in NTP mode 7 packets containing requests for information being sent to the server. These are **read–only** commands in that they make no modification of the server configuration state.

listpeers
: Obtain and print a brief list of the peers for which the server is maintaining state. These should include all configured peer associations as well as those peers whose stratum is such that they are considered by the server to be possible future synchronization candidates.

peers Obtain a list of peers for which the server is maintaining state, along with a summary of that state. Summary information includes the address of the remote peer, the local interface address (0.0.0.0 if a local address has yet to be determined), the stratum of the remote peer (a stratum of 16 indicates the remote peer is unsynchronized), the polling interval in seconds, the reachability register in octal, and the current estimated delay, offset and dispersion of the peer, all in seconds. In addition, the character in the left margin indicates the mode in which this peer entry is operating. A **+** denotes symmetric active, a – indicates symmetric passive, a **=** means the remote server is being polled in client mode, a **^** indicates that the server is broadcasting to this address, a **~** denotes that the remote peer is sending broadcasts, and a * marks the peer to which the server is currently synchronizing.

The contents of the host field may be one of four forms: host name, IP address, reference clock implementation name with its parameter, or **REFCLK(***implementation number*, *parameter*). On **hostnames no**, only IP–addresses are displayed.

dmpeers
: Show a slightly different peer summary list. Identical to the output of the **peers** command except for the character in the leftmost column. Characters only appear beside peers that were included in the final stage of the clock selection algorithm. A **.** indicates that this peer was cast off in the falseticker detection, while a **+** indicates that the peer made it through. A * denotes the peer with which the server is currently synchronizing.

showpeer *peer_address* [*addr2*] [*addr3*] [*addr4*]
: Show a detailed display of the current peer variables for one or more peers. Most of these values are described in the NTP Version 2 specification.

pstats *peer_address* [*addr2*] [*addr3*] [*addr4*]
: Show per–peer statistic counters associated with the specified peers.

loopinfo [**oneline | multiline**]
: Print the values of selected loop filter variables. The loop filter is the part of NTP that deals with adjustment of the local system clock. The **offset** is the last offset given to the loop filter by the packet-processing code. The **frequency** is actually the frequency error, or drift, of your system's clock in the units NTP uses for

internal computations. Dividing this number by 4096 should give you the actual drift rate. The **compliance** is actually a long-term average offset and is used by NTP to control the gain of the loop filter. The **timer** value is the number of seconds that have elapsed since a new sample offset was given to the loop filter. The **oneline** and **multiline** options specify the format in which this information is to be printed. **multiline** is the default.

sysinfo
Print a variety of system state variables, *i.e.*, the state related to the local server. Many of these values are described in the NTP Version 2 specification, RFC 1119.

sysstats
Print the number of stat counters maintained in the protocol module.

memstats
Print the number of counters related to the peer memory allocation code.

iostats Print counters maintained in the input–output module.

timerstats
Print counters maintained in the timer / event queue support code.

reslist Obtain and print the server's restriction list. This list is (usually) printed in sorted order and may help you understand how the restrictions are applied.

monlist
Obtain and print traffic counts collected and maintained by the monitor facility.

clockinfo *clock_peer_address* [*addr2*] [*addr3*] [*addr4*]
Obtain and print information concerning a peer clock. The values obtained provide information on the setting of fudge factors and other clock-performance information.

clkbug *clock_peer_address* [*addr2*] [*addr3*] [*addr4*]
Obtain debugging information for a clock peer. This information is provided only by some clock drivers and is mostly undecodable without a copy of the driver source in hand.

RUNTIME CONFIGURATION REQUESTS

All requests that cause state changes in the server are authenticated by the server using a configured NTP key (the facility can also be disabled by the server by not configuring a key). The key number and the corresponding key must also be made known to **xtnpdc**. To do so, use the **keyid** and **passwd** commands, the latter of which prompts for a password to use as the encryption key. You are also prompted automatically for both the key number and password the first time a command that would result in an authenticated request to the server is given. Authentication not only provides verification that the requester has permission to make such changes, but also gives an extra degree of protection again transmission errors.

Authenticated requests always include a timestamp in the packet data, which is included in the computation of the authentication code. This timestamp is compared by the server to its receive time stamp. If they differ by more than a small amount the request is rejected. This is done for two reasons. First, it makes simple replay attacks on the server, by someone who might be able to overhear traffic on your LAN, much more difficult. Second, it makes it more difficult to request configuration changes to your server from topologically remote hosts. While the reconfiguration facility works well with a server on the local host, and may work adequately between time–synchronized hosts on the same LAN, it works poorly for more distant hosts. As such, if reasonable passwords are chosen, care is taken in the distribution and protection of keys and appropriate source address restrictions are applied, the run time reconfiguration facility should provide an adequate level of security.

All of the following commands make authenticated requests:

addpeer *peer_address* [*keyid*] [*version#*] [**minpoll** | **prefer**]
: Add a configured, symmetric active peer association with a peer at the given address. If the optional *keyid* is a nonzero integer, all outgoing packets to the remote server have an authentication field attached encrypted with this key. If the value is 0 (or not given) no authentication is done. The *version#* can be 1 or 2, and defaults to 2. If **minpoll** is specified, the polling interval for the association remains clamped at the minimum. The latter option is only useful for testing. Note that an existing association with the same peer may be deleted when this command is executed, or may simply be converted to conform to the new configuration, as appropriate. The prefer keyword indicates a preferred peer (and thus is used primarily for clock synchronization, if possible). The preferred peer also determines the validity of the PPS signal – if the preferred peer is suitable for synchronization, so is the PPS signal.

addserver *peer_address* [*keyid*] [*version#*] [**minpoll** | **prefer**]
: Identical to the **addpeer** command, except that polling is done in client mode rather than symmetric active mode.

broadcast *peer_address* [*keyid*] [*version#*] [**minpoll**]
: Identical to the **addpeer** command except that packets are instead sent in broadcast mode. The **peer_address** parameter is generally a broadcast address on one of your local networks.

unconfig *peer_address* [*addr2*] [*addr3*] [*addr4*]
: Remove the configured bit from the specified peers. In many cases, this causes the peer association to be deleted. When appropriate, however, the association may persist in an unconfigured mode if the remote peer is willing to continue on in this fashion.

set bclient | auth [...]
: Allow the setting of the broadcast client and / or authenticate system flags. Setting the former causes the server to listen for broadcast NTP to to synchronize to broadcasts when appropriate. Setting the latter flag causes the server to only synchronize with peers that include an authentication field encrypted with one of the local server's trusted keys.

clear bclient | auth [...]
Allow the broadcast client and / or authenticate system flags to be cleared. Clearing the former causes incoming broadcast NTP packets to be ignored. Clearing the latter allows peers that have not included an authentication field, or that have included one, but have encrypted it with an untrusted key, to be considered synchronization candidates.

restrict *address mask flag* [*flag*]
Add flags to an existing restrict list entry, or add a new entry to the list with the specified flags. The possible choices for the flags arguments are:

ignore Ignore all packets from hosts that match this entry. If this flag is specified, neither queries nor time server polls are responded to.

noquery
Ignore all NTP mode 7 packets (*i.e.*, information queries and configuration requests) from the source. Time service is not affected.

nomodify
Ignore all NTP mode 7 packets that attempt to modify the state of the server (*i.e.*, run time reconfiguration). Queries that return information are permitted.

noserve
Ignore NTP packets whose mode is other than 7. In effect, time service is denied, though queries may still be permitted.

nopeer Provide stateless time service to polling hosts, but do not allocate peer memory resources to these hosts even if they otherwise might be considered useful as future synchronization partners.

notrust Treat these hosts normally in other respects, but never use them as synchronization sources.

ntpport
This is actually a match algorithm modifier, rather than a restriction flag. Its presence causes the restriction entry to be matched only if the source port in the packet is the standard NTP UDP port (123). Both **ntpport** and non–**ntpport** may be specified. The **ntpport** is considered more specific and is sorted later in the list.

unrestrict *address mask flag* [*flag*]
Remove the specified flags from the restrict list entry indicated by the *address* and *mask* arguments.

delrestrict *address mask* [**ntpport**]
Delete the matching entry from the restrict list.

monitor yes | no
Enable or disable the monitoring facility. Note that a **monitor no** command followed by a **monitor yes** command is a good way of resetting the packet counts.

readkeys
Purge the current set of authentication keys and obtain a new set by rereading

the keys file (which must have been specified in the **xntpd**(1M) configuration file). This option allows encryption keys to be changed without restarting the server.

trustkey *keyid* [*keyid*] [*keyid*] [*keyid*]
: Add one or more keys to the trusted key list. When authentication is enabled, peers whose time is to be trusted must be authenticated using a trusted key.

untrustkey *keyid* [*keyid*] [*keyid*] [*keyid*]
: Remove one or more keys from the trusted key list.

authinfo
: Return information concerning the authentication module, including known keys and counts of encryptions and decryptions that have been done.

setprecision *precision_value*
: Set the precision that the server advertises to the specified value. This should be a negative integer in the range -4 through -20.

setselect *algorithm_number*
: Set the selection weight algorithm to that indicated by the specified number. The number should be an integer value between 1 and 5, inclusive. Algorithm 1 is that specified in RFC 1119, the other 4 algorithms are experimental; use them with caution.

SEE ALSO

xntpd(1M)